

## New Algorithms for Verifying the Null Space Condition in Compressed Sensing

MYUNG CHO\*,

*Dept. of ECE, University of Iowa, Iowa City, IA, 52242*

\*myung-cho@uiowa.edu

KUMAR VIJAY MISHRA

*Dept. of ECE, University of Iowa, Iowa City, IA, 52242*

kumarvijay-mishra@uiowa.edu

AND

WEIYU XU

*Dept. of ECE, University of Iowa, Iowa City, IA, 52242*

weiyu-xu@uiowa.edu

The null space condition for  $\ell_1$  minimization in compressed sensing is a necessary and sufficient condition on the sensing matrices under which a sparse signal can be uniquely recovered from the observation data via  $\ell_1$  minimization. However, verifying the null space condition is known to be computationally challenging. Most of the existing methods can provide only upper and lower bounds on the proportion parameter that characterizes the null space condition. In this paper, we propose new polynomial-time algorithms to establish the upper bounds of the proportion parameter. Based on these polynomial-time algorithms, we have designed new algorithms - Sandwiching Algorithm (SWA) and Tree Search Algorithm (TSA) - to precisely verify the null space condition. Simulation results show that our polynomial-time algorithms can achieve better bounds on recoverable sparsity with low computational complexity than existing methods in the literature. We also show that Tree Search Algorithm and Sandwiching Algorithm significantly reduce the computational complexity when compared with the exhaustive search method.

*Keywords:* compressed sensing, null space condition,  $\ell_1$  minimization, performance guarantee

### 1. Introduction

Compressed sensing is an efficient signal processing technique to recover a sparse signal from fewer samples than required by the Nyquist-Shannon theorem, greatly reducing the time and energy spent in sampling operation. These advantages make compressed sensing attractive for various applications such as medical imaging [20, 21, 17, 13], face recognition [32, 24, 30], radar [11], astronomy [1] and holography [2, 22, 25].

In compressed sensing, we are interested in recovering the sparsest vector  $x \in \mathbb{R}^n$  that satisfies the underdetermined system  $y = Ax$ . Here  $\mathbb{R}$  is the set of real numbers,  $A \in \mathbb{R}^{m \times n}$ ,  $m < n$ , is the sensing matrix, and  $y \in \mathbb{R}^m$  is the observation or measurement data. This can be posed as an  $\ell_0$  minimization problem:

$$\begin{aligned} & \text{minimize} && \|x\|_0 \\ & \text{subject to} && Ax = y, \end{aligned} \tag{1.1}$$

where  $\|x\|_0$  is the number of non-zero elements in vector  $x$ . The  $\ell_0$  minimization is an NP-hard problem

and, therefore, it is often solved by relaxing to its closest convex approximation - the  $\ell_1$  minimization problem:

$$\begin{aligned} & \text{minimize} \quad \|x\|_1 \\ & \text{subject to} \quad Ax = y. \end{aligned} \quad (1.2)$$

It has been shown that the optimal solution of  $\ell_0$  minimization can be obtained by solving  $\ell_1$  minimization under certain conditions (e.g. Restricted Isometry Property or RIP) [5, 9, 3, 4, 10]. For random sensing matrices, these conditions hold with high probability. We note that the RIP is a sufficient condition for sparse recovery [27].

A necessary and sufficient condition under which a  $k$ -sparse signal  $x$ , ( $k \ll n$ ) can be uniquely obtained via  $\ell_1$  minimization is the null space condition (NSC) [16, 8, 9]. A matrix  $A$  satisfies the NSC for a positive integer  $k$  if

$$\|z_K\|_1 < \|z_{\bar{K}}\|_1, \quad (1.3)$$

holds true for all  $z \in \{z \mid Az = 0, z \neq 0\}$  and for all subsets  $K \subseteq \{1, 2, \dots, n\}$  with  $|K| \leq k$ . Here  $K$  is an index set,  $|K|$  is the cardinality of  $K$ ,  $z_K$  is the part of the vector  $z$  over the index set  $K$ , and  $\bar{K}$  is the complement of  $K$ . The null space condition is related to the proportion parameter  $\alpha_k$  defined as

$$\alpha_k = \max_{\{z: Az=0, z \neq 0\}} \max_{\{K: |K| \leq k\}} \frac{\|z_K\|_1}{\|z\|_1}. \quad (1.4)$$

The  $\alpha_k$  is the optimal value of the following optimization problem:

$$\begin{aligned} & \max_{z, K} \quad \|z_K\|_1 \\ & \text{subject to} \quad \|z\|_1 \leq 1, \\ & \quad \quad \quad Az = 0, \end{aligned} \quad (1.5)$$

where  $K$  is a subset of  $\{1, 2, \dots, n\}$  with cardinality at most  $k$ . The matrix  $A$  satisfies the null space condition for a positive integer  $k$  if and only if  $\alpha_k < \frac{1}{2}$ . Equivalently, the null space condition can be verified by computing or estimating  $\alpha_k$ . The role of  $\alpha_k$  is also important in recovery of an approximately sparse signal  $x$  via  $\ell_1$  minimization where a smaller  $\alpha_k$  implies more robustness [16, 8, 33].

In this paper, we are interested in computing  $\alpha_k$  and, in particular, finding the maximum  $k$  for which  $\alpha_k < \frac{1}{2}$ . However, computing  $\alpha_k$  to verify the null space condition is extremely expensive and was reported in [27] to be strongly NP-hard. Due to the challenges in computing  $\alpha_k$ , verifying the null space condition explicitly for deterministic sensing matrices remains a relatively unexamined research area in the existing literature. In [18, 26, 9, 16], convex relaxations were used to establish the upper or lower bounds of  $\alpha_k$  (or other parameters related to  $\alpha_k$ ) instead of computing the exact  $\alpha_k$ . While [9, 18] proposed semidefinite programming based methods, [16, 26] suggested linear programming relaxations to determine the upper and lower bounds of  $\alpha_k$ . For both methods, computable performance guarantees were reported on sparse signal recovery via bounding  $\alpha_k$ . However, these relaxation bounds of  $\alpha_k$  could only verify the null space condition with  $k = O(\sqrt{n})$ , even though theoretically  $k$  can grow linearly with  $n$ .

In this work, we propose polynomial-time algorithms, called pick- $l$ -element algorithms ( $l \geq 1$ ) which compute upper bounds of  $\alpha_k$ . We then use pick- $l$ -element algorithms to develop new methods - Sandwiching Algorithm (SWA) and Tree Search Algorithm (TSA) - to compute the *exact*  $\alpha_k$  by significantly reducing computational complexity of an exhaustive search method. These algorithms offer ways to control a smooth tradeoff between complexity and accuracy of the computations.

Our simulations show that for large-dimensional sensing matrices ( $n \geq 1024$ ), our pick-1-element algorithm and early-terminated TSA can achieve better bounds on recoverable sparsity, and shorter computational time, compared with methods that use linear programming (LP) [16] and semidefinite programming (SDP) [9]. We have verified the recoverable sparsity  $k$  using pick-1-element algorithm up to 500. This is much larger than bounds on recoverable sparsity obtained through the LP and SDP methods. For low-dimensional sensing matrices ( $n$  up to a few hundred), our pick- $l$ -element ( $l \leq 3$ ) algorithm also achieves larger recoverable sparsity (or better bounds on  $\alpha_k$ ) than the LP and SDP methods. For small sensing matrices ( $n$  up to around 100), SWA and TSA can give even precise bounds on  $\alpha_k$  for  $k$  up to 5, with much smaller complexity than the exhaustive search method. By computing the exact  $\alpha_k$  using SWA and TSA, we obtained bigger recoverable  $k$  values than the methods from [9] and [16]. We also provide an interesting network tomography application [19, 34, 15] where our algorithms can be used to verify the null space condition for sensing matrices. We demonstrate the effectiveness of our algorithms for a few selected networks (graphs) through numerical experiments.

### 1.1 Notations and Preliminaries

We denote the sets of real numbers, and positive integers as  $\mathbb{R}$  and  $\mathbb{Z}^+$  respectively. We reserve uppercase letters  $K$  and  $L$  for index sets, and lowercase letters  $k, l \in \mathbb{Z}^+$  for their respective cardinalities. We sometimes use  $|\cdot|$  to denote the cardinality of a set. We assume  $k > l > 0$  throughout the paper. For vectors or scalars, we use lowercase letters, e.g.,  $x, k, l$ . For a vector  $x \in \mathbb{R}^n$ , we use  $x_i$  for its  $i$ -th element. We use  $J^{(i)}$  to denote the  $i$ -th element of a set  $J$ . Since the number of columns of a sensing matrix  $A$  is  $n$ , the full index set is  $\{1, 2, \dots, n\}$ . In addition, we represent the  $\binom{n}{l}$  subsets of the full index set with  $L_i$ , where  $i = 1, 2, \dots, \binom{n}{l}$  and  $|L_i| = l$ . We use “\*” superscript, such as  $(z^*, K^*)$ , to represent an optimal solution of an optimization problem. The maximum value of  $k$  such that both  $\alpha_k < \frac{1}{2}$  and  $\alpha_{k+1} \geq \frac{1}{2}$  hold true is denoted by the *maximum recoverable sparsity*  $k_{\max}$ .

## 2. Pick- $l$ -element Algorithm

Consider a sensing matrix with  $n$  columns. Then, there are  $\binom{n}{k}$  subsets  $K$  each of cardinality  $k$ . When  $n$  and  $k$  are large, combinatorial search over these subsets to compute  $\alpha_k$  could be very arduous. For example, when  $n = 100$  and  $k = 10$ , it would take a search over  $1.7310\text{e}+13$  subsets to compute  $\alpha_k$  - a combinatorial task that is beyond the technological reach of common desktop computers. Our goal is to devise algorithms that can rapidly yield the exact value of  $\alpha_k$ . As an initial step, we develop a method to compute an upper bound of  $\alpha_k$  in polynomial time. We refer this method as the *pick- $l$ -element algorithm*; initially, it picks a positive integer  $l$  and then solves the following optimization problem for each of the possible (fixed) index set  $L$  for which  $|L| = l$ :

$$\begin{aligned} & \underset{z}{\text{maximize}} && \|z_L\|_1 \\ & \text{subject to} && \|z\|_1 \leq 1, \\ & && Az = 0. \end{aligned} \tag{2.1}$$

We remark that  $z$  is in the null space of  $A$ . We can also change (2.1) to a formulation that involves the basis of the null space of  $A$ . Note that while (1.5) is optimized over all the index sets  $K$  with  $|K| = k$ , the index set  $L$ ,  $|L| = l < k$  is fixed for the optimization routine in (2.1). For a given index set  $L$ , we define the proportion parameter as follows:

$$\alpha_{l,L} = \underset{\{z: Az=0, z \neq 0\}}{\text{maximize}} \frac{\|z_L\|_1}{\|z\|_1}. \quad (2.2)$$

The pick- $l$ -element algorithm uses  $\alpha_{l,L}$ 's obtained from different index sets to compute an upper bound of  $\alpha_k$ . For small values of  $l$ , e.g.,  $l = 1, 2, 3$ , the computation of  $\alpha_{l,L}$  for all the index sets  $L$  requires much less effort than directly obtaining  $\alpha_k$  because the number of operations is reduced from  $\binom{n}{k}$  enumerations to  $\binom{n}{l}$ . With this reduced enumerations, we still have meaningful upper bounds of  $\alpha_k$ . Algorithm 1 lists all the steps of the pick- $l$ -element algorithm. Finally, we provide a useful result in Lemma 2.1 to derive the upper bound of the proportion parameter for a fixed index set  $K$ , and then, we show that the pick- $l$ -element algorithm yields an upper bound of  $\alpha_k$  in Lemma 2.2. Since we are also interested in  $k_{\max}$ , we bound this quantity in Lemma 2.3.

LEMMA 2.1 ('Cheap' upper bound for a given subset  $K$ ) Given a subset  $K$ , we have

$$\alpha_{k,K} \leq \frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K, |L_i|=l\}} \alpha_{l,L_i}. \quad (2.3)$$

*Proof.* Suppose  $\frac{\|z_{L_i}\|_1}{\|z\|_1}$  achieves the maximum value when  $z = z^i$ ,  $i = 1, \dots, \binom{k}{l}$ , and  $\frac{\|z_K\|_1}{\|z\|_1}$  achieves the maximum value when  $z = z^*$ . Since each element of  $K$  appears in  $\binom{k-1}{l-1}$  subsets  $L_i$  such that  $L_i \subseteq K$  and  $|L_i| = l$ , we obtain the following inequality:

$$\alpha_{k,K} = \frac{\|z_K^*\|_1}{\|z^*\|_1} = \frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K, |L_i|=l\}} \frac{\|z_{L_i}^*\|_1}{\|z^*\|_1} \leq \frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K, |L_i|=l\}} \frac{\|z_{L_i}^i\|_1}{\|z^i\|_1}.$$

□

LEMMA 2.2 The pick- $l$ -element algorithm provides upper bounds of  $\alpha_k$ , namely

$$\alpha_k \leq \frac{1}{\binom{k-1}{l-1}} \sum_{i=1}^{\binom{k}{l}} \alpha_{l,L_i}, \quad (2.4)$$

$$\text{where } \alpha_{l,L_1} \geq \alpha_{l,L_2} \geq \dots \geq \alpha_{l,L_i} \geq \dots \geq \alpha_{l,L_{\binom{k}{l}}}. \quad (2.5)$$

*Proof.* Without loss of generality, we assume that when  $z = z^i$ ,  $i = 1, 2, \dots, \binom{n}{l}$ , the optimal values  $\alpha_{l,L_i}$ 's from (2.1) are obtained and  $\alpha_{l,L_i}$ 's are sorted in descending order. It is noteworthy that  $\alpha_{k,K}$  is defined for a fixed  $K$  set, but  $\alpha_k$  is the maximum value over all the subsets with cardinality  $k$ . Recall that the superscript “\*” is used for an optimal solution (e.g.  $z^*, K^*$ ). From the aforementioned definitions and similar argument as in Lemma 2.1, we have:

$$\alpha_k = \alpha_{k,K^*} \leq \frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K^*, |L_i|=l\}} \frac{\|z_{L_i}^i\|_1}{\|z^i\|_1} \leq \frac{1}{\binom{k-1}{l-1}} \sum_{i=1}^{\binom{k}{l}} \frac{\|z_{L_i}^i\|_1}{\|z^i\|_1} = \frac{1}{\binom{k-1}{l-1}} \sum_{i=1}^{\binom{k}{l}} \alpha_{l,L_i}.$$

The last inequality is obtained from the assumption that  $\alpha_{l,L_i}$  are sorted in descending order. □

LEMMA 2.3 The maximum recoverable sparsity  $k_{\max}$  satisfies

$$k_{\max} \geq \left\lceil l \cdot \frac{1/2}{\alpha_l} \right\rceil - 1, \quad (2.6)$$

where  $\lceil \cdot \rceil$  is the ceiling function.

---

**Algorithm 1** Pick- $l$ -element Algorithm,  $1 \leq l < k$  for computing upper bounds of  $\alpha_k$ 


---

- 1: Given a matrix  $A$ , solve (2.1) for all the subsets  $L$ , such that  $|L| = l : \alpha_{l,L}$  for all the subsets  $L$ .
- 2: Sort these  $\binom{n}{l}$  different values of  $\alpha_{l,L}$  in descending order :  $\alpha_{l,L_i}$ , where  $i = 1, 2, \dots, \binom{n}{l}$ . ( $\alpha_{l,L_i} \geq \alpha_{l,L_j}$  when  $i \leq j$ .)
- 3: Compute the upper bound of  $\alpha_k$  by summing up the first  $\binom{k}{l}$  values of  $\alpha_{l,L_i}$  and divide it by  $\binom{k-1}{l-1}$ :

$$\overline{\alpha}_k \triangleq \frac{1}{\binom{k-1}{l-1}} \sum_{i=1}^{\binom{k}{l}} \alpha_{l,L_i} \quad (2.8)$$

- 4: If  $\overline{\alpha}_k < \frac{1}{2}$ , then the null space condition for the number  $k$  is satisfied.
- 

*Proof.* To prove this lemma, we will show that when  $k = \lceil l \cdot \frac{1/2}{\alpha_l} \rceil - 1$ ,  $\alpha_k < \frac{1}{2}$ . This can be concluded from an upper bound of  $\alpha_k$  given as follows:

$$\alpha_k = \alpha_{k,K^*} \leq \frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K^*, |L_i|=l\}} \alpha_{l,L_i} \leq \frac{\binom{k}{l}}{\binom{k-1}{l-1}} \alpha_l = \alpha_l \cdot \frac{k}{l}. \quad (2.7)$$

Note that  $\binom{k}{l}$  terms exist in the summation. From (2.7), if  $\alpha_l \cdot \frac{k}{l} < \frac{1}{2}$ , then  $\alpha_k < \frac{1}{2}$ . In other words, if  $k < l \cdot \frac{1/2}{\alpha_l}$ , then  $\alpha_k < \frac{1}{2}$ . Since  $k$  is a positive integer, when  $k = \lceil l \cdot \frac{1/2}{\alpha_l} \rceil - 1$ ,  $\alpha_k < \frac{1}{2}$ . Therefore, the maximum recoverable sparsity  $k_{max}$  should be larger than or at least equal to  $\lceil l \cdot \frac{1/2}{\alpha_l} \rceil - 1$ .  $\square$

### 2.1 Calculating $\alpha_{l,L}$ for a given subset $L$ and incorporating sign patterns

It is not trivial to solve the optimization problem in (2.1) since it maximizes a convex function. In order to solve (2.1) for a given subset  $L$ , we cast (2.1) as  $2^l$  linear programming problems by considering all the possible sign patterns (+1 or -1) of every element of  $z_L$  (e.g., if  $l = 2$  so that  $z_L = [z_1, z_2]$ , then,  $\|z_L\|_1 = |z_1| + |z_2|$  can correspond to  $2^l = 4$  possibilities:  $[z_1, z_2]$ ,  $[z_1, -z_2]$ ,  $[-z_1, z_2]$  and  $[-z_1, -z_2]$ ).  $\alpha_{l,L}$  is equal to the maximum among the  $2^l$  objective values.

We can introduce the concept of signed subset  $L_{\text{sign}}$ , whose elements are positive or negative indices (for example,  $L_{\text{sign}} = \{-1, 2, -5, -7\}$ ).  $\alpha_{l,L_{\text{sign}}}$  is defined as the optimal value of the following optimization problem:

$$\begin{aligned} & \underset{z}{\text{maximize}} && \langle \text{sign}(L_{\text{sign}}), z_{L_{\text{abs}}} \rangle \\ & \text{subject to} && \|z\|_1 \leq 1, Az = 0 \\ & && \text{sign}(i) \cdot z_{|i|} \geq 0, i \in L_{\text{sign}} \end{aligned} \quad (2.9)$$

where  $\langle \cdot, \cdot \rangle$  is the inner product,  $L_{\text{sign}}$  is a set of nonzero integers,  $\text{sign}(z_{L_{\text{sign}}})$  is a vector that consists of  $\pm 1$ , and  $L_{\text{abs}}$  is a set of absolute values of elements of  $L_{\text{sign}}$  (e.g.,  $L_{\text{sign}} = \{-1, 2, -5, -7\}$ ,  $L_{\text{abs}} = \{1, 2, 5, 7\}$ ,  $\text{sign}(z_{L_{\text{sign}}}) = [-1, 1, -1, -1]$  and  $z_{L_{\text{abs}}} = [z_1, z_2, z_5, z_7]$ .) Thus  $\alpha_{l,L}$  is the maximum among all the  $2^l$   $\alpha_{l,L_{\text{sign}}}$ 's. In fact, to obtain  $\alpha_{l,L}$ , we only need to compute  $2^{l-1}$   $\alpha_{l,L_{\text{sign}}}$ 's instead of  $2^l$ . This is because  $\alpha_{l,L_{\text{sign}}}$  (e.g.,  $\alpha_{2,\{1, -2\}}$ ) equal to its complementary case (e.g.,  $\alpha_{2,\{-1, 2\}}$ ), due to symmetry in (2.9).

### 3. Optimized pick- $l$ -element algorithm

We can tighten the bounds of  $\alpha_k$  in the pick- $l$ -element algorithm by replacing the constant factor  $\frac{1}{\binom{k-1}{l-1}}$  with optimized coefficients at the cost of additional complexity. The theoretical merit of this *optimized* pick- $l$ -element algorithm lies in the fact that as  $l$  increases, the upper bound of  $\alpha_k$  becomes smaller or stays the same.

The optimized pick- $l$ -element algorithm provides the upper bound of  $\alpha_k$  through the optimal value of the following optimization problem:

$$\begin{aligned}
 & \text{maximize} && \sum_{i=1}^{\binom{n}{l}} \gamma_i \alpha_{l,L_i} \\
 & \text{subject to} && \gamma_i \geq 0, \quad 1 \leq i \leq \binom{n}{l}, \\
 & && \sum_{i=1}^{\binom{n}{l}} \gamma_i \leq \frac{k}{l}, \\
 & && \sum_{\{i: B \subseteq L_i, 1 \leq i \leq \binom{n}{l}\}} \gamma_i \leq \frac{\binom{k-b}{l-b}}{\binom{k-1}{l-1}}, \quad \begin{array}{l} \text{for all integers } b \text{ such that } 1 \leq b \leq l, \\ \text{for all subsets } B \text{ with } |B| = b \end{array}.
 \end{aligned} \tag{3.1}$$

In the following lemmata, we show that the optimized pick- $l$ -element algorithm produces an upper bound of  $\alpha_k$ , and that this bound is tighter than the upper bound obtained from the basic pick- $l$ -element algorithm described in Section 2. The last lemma establishes that as  $l$  increases, the upper bound of  $\alpha_k$  decreases or remains unchanged.

LEMMA 3.1 The pick- $l$ -element algorithm with optimized coefficients provides upper bounds of  $\alpha_k$ .

*Proof.* The strategy to prove Lemma 3.1 is to show that one feasible solution of (3.1) gives no smaller objective value than  $\alpha_k$ . Let us assume that  $\gamma_i = \frac{1}{\binom{k-1}{l-1}}$  when  $L_i \subseteq K^*$ , and  $\gamma_i = 0$  otherwise. This assumption yields a feasible solution of (3.1). From this feasible solution, the optimal value of (3.1) is bigger than or at least equal to  $\frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K^*, |L_i|=l\}} \alpha_{l,L_i}$ . With this objective value, we can obtain the following inequality which we have proven in Lemma 2.1:

$$\alpha_k = \alpha_{k,K^*} \leq \frac{1}{\binom{k-1}{l-1}} \sum_{\{i: L_i \subseteq K^*, |L_i|=l\}} \alpha_{l,L_i} \tag{3.2}$$

Note that the sum of the coefficients in (3.2) is  $\frac{1}{\binom{k-1}{l-1}} \times \binom{k}{l} = \frac{k}{l}$  - the second constraint in (3.1). For the set  $K^*$  with cardinality  $k$ , and any subset  $B \subseteq K^*$  with cardinality  $|B| = b$ , there are  $\binom{k-b}{l-b}$  subsets  $L$  with cardinality  $l$ , such that  $B \subseteq L \subseteq K^*$ . For each subset  $L$ , the corresponding  $\gamma_i = \frac{1}{\binom{k-1}{l-1}}$ , and thus the summation of those  $\gamma_i$ 's over  $\binom{k-b}{l-b}$  such subsets of cardinality  $l$  will be equal to  $\frac{\binom{k-b}{l-b}}{\binom{k-1}{l-1}}$ . Hence, the coefficients  $\frac{1}{\binom{k-1}{l-1}}$  in (3.2) satisfy the third constraints of (3.1).  $\square$

LEMMA 3.2 The optimized pick- $l$ -element algorithm provides a tighter, or at least the same, upper bound of  $\alpha_k$  than the basic pick- $l$ -element algorithm.

*Proof.* We can easily see that the following optimization problem (3.3) provides the same result as that from the basic pick- $l$ -element algorithm:

$$\begin{aligned}
& \underset{\gamma_i, 1 \leq i \leq \binom{n}{l}}{\text{maximize}} && \sum_{i=1}^{\binom{n}{l}} \gamma_i \alpha_{l, L_i} \\
& \text{subject to} && 0 \leq \gamma_i \leq \frac{1}{\binom{k-1}{l-1}}, \quad 1 \leq i \leq \binom{n}{l}, \\
& && \sum_{i=1}^{\binom{n}{l}} \gamma_i \leq \frac{k}{l}.
\end{aligned} \tag{3.3}$$

The optimization problem (3.3) is a relaxation of (3.1). Therefore, the optimized pick- $l$ -element algorithm provides a tighter, or at least the same, upper bound than the basic pick- $l$ -element algorithm.  $\square$

**LEMMA 3.3** The optimized pick- $l$ -element algorithm provides tighter, or at least the same, upper bounds than the optimized pick- $p$ -element algorithm when  $l > p$ .

*Proof.* From Lemma 2.1, we can upper bound the result obtained from (3.1) by that of the following program:

$$\begin{aligned}
& \underset{\gamma_i, 1 \leq i \leq \binom{n}{l}}{\text{maximize}} && \frac{1}{\binom{l-1}{p-1}} \sum_{i=1}^{\binom{n}{l}} \gamma_i \sum_{\{j: P_j \subset L_i, |P_j|=p\}} \alpha_{p, P_j} \\
& \text{subject to} && \gamma_i \geq 0, \quad 1 \leq i \leq \binom{n}{l}, \\
& && \sum_{i=1}^{\binom{n}{l}} \gamma_i \leq \frac{k}{l}, \\
& && \sum_{\{i: B \subseteq L_i, 1 \leq i \leq \binom{n}{l}\}} \gamma_i \leq \frac{\binom{k-b}{l-b}}{\binom{k-1}{l-1}}, \quad \begin{array}{l} \text{for all integers } b \text{ such that } 1 \leq b \leq l, \\ \text{for all subsets } B \text{ with } |B| = b \end{array}.
\end{aligned} \tag{3.4}$$

Note that in the objective function of (3.4), each  $\alpha_{p, P_j}$ ,  $1 \leq j \leq \binom{n}{p}$  appears  $\binom{n-p}{l-p}$  times. Let  $\gamma'_j = \frac{1}{\binom{l-1}{p-1}} \sum_{\{i: P_j \subset L_i, 1 \leq i \leq \binom{n}{l}\}} \gamma_i$  and relax (3.4) to obtain the following optimization problem (3.5), which turns out to be the same as the optimized pick- $p$ -element algorithm.

$$\begin{aligned}
& \underset{\gamma'_j, 1 \leq j \leq \binom{n}{p}}{\text{maximize}} && \sum_{j=1}^{\binom{n}{p}} \gamma'_j \alpha_{p, P_j} \\
& \text{subject to} && \gamma'_j \geq 0, \quad 1 \leq j \leq \binom{n}{p}, \\
& && \sum_{j=1}^{\binom{n}{p}} \gamma'_j \leq \frac{k}{p}, \\
& && \sum_{\{j: I \subseteq P_j, 1 \leq j \leq \binom{n}{p}\}} \gamma'_j \leq \frac{\binom{k-b}{p-b}}{\binom{k-1}{p-1}}, \quad \begin{array}{l} \text{for all integers } b \text{ such that } 1 \leq b \leq p, \\ \text{for all subsets } I \text{ with } |I| = b \end{array}.
\end{aligned} \tag{3.5}$$

The constraints of (3.5) can be obtained from the relaxations of the constraints of (3.4). The first constraint of (3.5) is trivial to obtain. From the following simplification, we can obtain the second constraint of (3.5):

$$\sum_{j=1}^{\binom{n}{p}} \gamma_j' = \sum_{j=1}^{\binom{n}{p}} \frac{1}{\binom{l-1}{p-1}} \sum_{\{i: P_j \subset L_i, 1 \leq i \leq \binom{n}{l}\}} \gamma_i = \frac{1}{\binom{l-1}{p-1}} \binom{l}{p} \sum_{i=1}^{\binom{n}{l}} \gamma_i \leq \frac{1}{\binom{l-1}{p-1}} \binom{l}{p} \frac{k}{l} = \frac{k}{p}.$$

The third constraint in (3.5) can be deduced from the following inequality:

$$\begin{aligned} \sum_{\{j: B \subseteq P_j, 1 \leq j \leq \binom{n}{p}, |B|=b\}} \gamma_j' &= \sum_{\{j: B \subseteq P_j, 1 \leq j \leq \binom{n}{p}, |B|=b\}} \frac{1}{\binom{l-1}{p-1}} \sum_{\{i: P_j \subset L_i, 1 \leq i \leq \binom{n}{l}\}} \gamma_i \\ &= \frac{1}{\binom{l-1}{p-1}} \frac{\binom{n-b}{p-b} \binom{n-p}{l-p}}{\binom{n-b}{l-b}} \sum_{\{i: B \subset L_i, 1 \leq i \leq \binom{n}{l}, |B|=b\}} \gamma_i \\ &\leq \frac{1}{\binom{l-1}{p-1}} \frac{\binom{n-b}{p-b} \binom{n-p}{l-p}}{\binom{n-b}{l-b}} \frac{\binom{k-b}{l-b}}{\binom{k-1}{l-1}}, \quad 1 \leq b \leq p \\ &= \frac{\binom{k-b}{p-b}}{\binom{k-1}{p-1}}, \quad 1 \leq b \leq p \end{aligned}$$

Since (3.5) is obtained from a relaxation of (3.4), the optimal value of (3.5) is larger or equal to the optimal value of (3.4). The (3.5) is just the optimized pick- $p$ -element algorithm. Therefore, when  $l > p$ , the optimized pick- $l$ -element algorithm provides tighter or at least the same upper bounds than the optimized pick- $p$ -element algorithm.  $\square$

#### 4. Sandwiching Algorithm

We obtained upper bounds of  $\alpha_k$  through the basic or optimized pick- $l$ -element algorithm in Sections 2 and 3 respectively. However, these algorithms do not provide an *exact* value of  $\alpha_k$ . In order to obtain an exact  $\alpha_k$  value, we devise the Sandwiching Algorithm (SWA), which greatly reduces the computational complexity compared to the exhaustive search method to find the exact  $\alpha_k$ . We remark that the convex programming methods in [9] and [16] only provide upper bounds of  $\alpha_k$ , instead of an exact value of  $\alpha_k$  except when  $k = 1$ .

The idea of SWA is to maintain upper and lower bounds in computing an exact value of  $\alpha_k$ . In algorithm execution, we constantly decrease the upper bound and increase the lower bound, noting that the exact  $\alpha_k$  is *sandwiched* between these two bounds (referred later as the global upper bound and the global lower bound respectively). When the lower bound and the upper bound meet, we obtain a certificate that the exact value of  $\alpha_k$  has been found.

There are two ways to obtain the upper bound of  $\alpha_k$ : ‘cheap’ upper bound and linear programming based upper bound. We introduced the ‘cheap’ upper bound in Lemma 2.1. We now define the linear programming based upper bound in Lemma 4.1. Further, in Lemma 4.2, we show that this upper bound is not larger than the cheap upper bound.

**LEMMA 4.1** (Linear programming based upper bound for a given index set  $K$ ) An upper bound of  $\alpha_{k,K}$  for a given index set  $K$  can be obtained by solving the following optimization problem (4.1):

$$\begin{aligned} &\text{maximize} \quad \sum_{j \in K} b_j \\ &\text{subject to} \quad \sum_{t \in L_i} b_t \leq \alpha_{l,L_i}, \text{ for all } L_i \subseteq K, i = 1, \dots, \binom{k}{l}. \\ &\quad b_j \geq 0, \quad j = 1, 2, \dots, k. \end{aligned} \tag{4.1}$$



*Proof.* From the definition of  $\alpha_{k,K}$ , we can rewrite  $\alpha_{k,K}$  as the optimal value of the following optimization problem:

$$\begin{aligned} & \underset{z \in \mathbb{R}^n}{\text{maximize}} && \frac{\|z_K\|_1}{\|z\|_1} \\ & \text{subject to} && \frac{\|z_{L_i}\|_1}{\|z\|_1} \leq \alpha_{l,L_i}, \quad i = 1, \dots, \binom{k}{l}, \\ & && Az = 0. \end{aligned} \tag{4.2}$$

Since  $\alpha_{l,L_i}$  is the maximum value, the newly added constraints  $\frac{\|z_{L_i}\|_1}{\|z\|_1} \leq \alpha_{l,L_i}$  are just redundant, which always hold true over  $z$  such that  $Az = 0$ . Representing  $b_t = \frac{\|z_{\{t\}}\|_1}{\|z\|_1}$ ,  $t = 1, \dots, n$ , we can relax (4.2) to (4.1). Thus, the optimal value of (4.1) is an upper bound on that of (4.2), namely  $\alpha_{k,K}$ .  $\square$

LEMMA 4.2 The linear programming based upper bound via (4.1) is not larger than the ‘cheap’ upper bound via (2.3).

*Proof.* Summing up the first constraints in (4.1), we get

$$\sum_{i=1}^{\binom{k}{l}} \sum_{t \in L_i} b_t \leq \sum_{i=1}^{\binom{k}{l}} \alpha_{l,L_i},$$

where  $L_i$  are subsets of a given subset  $K$ . Since each element in a subset  $K$  appears in  $\binom{k-1}{l-1}$  subsets with  $|L_i| = l$ , we get

$$\binom{k-1}{l-1} \sum_{t \in K} b_t \leq \sum_{i=1}^{\binom{k}{l}} \alpha_{l,L_i}. \tag{4.3}$$

Finally, we obtain (2.3) from (4.3).  $\square$

#### 4.1 Algorithm

SWA consists of three major steps (see Algorithm 2). In the first step, SWA computes  $\alpha_{l,L}$  for all the subsets  $L$  with a fixed cardinality  $l$ . The second step involves calculating cheap upper bounds (2.3) of  $\alpha_{k,K}$  for all the index sets  $K$  using  $\alpha_{l,L}$  values, and sorts the  $\binom{n}{k}$  index sets  $K$  in descending order of the upper bounds of  $\alpha_{k,K}$ . Finally, SWA computes the exact  $\alpha_{k,K}$  for each index set  $K$ , starting at the top of the sorted list, until the upper bound coincides with the lower bound in the algorithm. In this step, linear programming based upper bounds of  $\alpha_{k,K}$  for some index sets  $K$  are computed for a tighter upper bound of  $\alpha_{k,K}$  than a cheap upper bound.

At any point in the execution of SWA, the global upper bound is the tightest upper bound of  $\alpha_k$ ; and the global lower bound is the largest  $\alpha_{k,K}$  among all the index sets  $K$  for which the exact  $\alpha_{k,K}$  has been computed. Since  $\alpha_k$  is the largest  $\alpha_{k,K}$  among all the index sets  $K$ , the global lower bound is always lower or at most equal to  $\alpha_k$ . The following theorem claims that Algorithm 2 will yield the exact value of  $\alpha_k$  in a finite number of steps.

THEOREM 4.1 Both the global lower and the global upper bound of  $\alpha_k$  converge to  $\alpha_k$  in a finite number of steps.

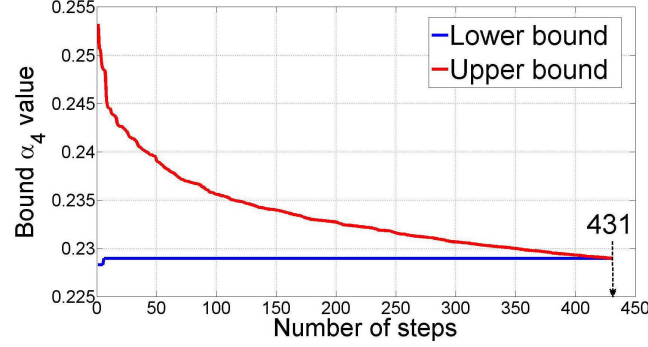


FIG. 1: An illustration of sandwiching algorithm on  $70 \times 80$  Gaussian sensing matrix  $A$  for  $(k, l) = (4, 2)$ . The exhaustive search goes through all  $\binom{80}{4} \approx 1.58e6$  possible index sets  $K$ . The sandwiching algorithm searches 431 index sets.

*Proof.* In SWA, we use the pick- $l$ -element algorithm to calculate the values of  $\alpha_{i,L}$  for every subset  $L$  as a first step. By using the cheap upper bound (2.3), we calculate upper bounds of  $\alpha_{k,K}$  for every index set  $K$ . We then sort these index sets in descending order of their upper bounds.

In algorithm execution, these sorted index sets are visited one-by-one and the exact  $\alpha_{k,K}$  values for the given index sets are calculated. The global upper bound (GUB) of  $\alpha_k$ , which is the biggest upper bound among index sets over which exact  $\alpha_{k,K}$  has not been calculated yet, never increases due to sorting. In the meanwhile, the global lower bound (GLB), which is so far the largest exact  $\alpha_{k,K}$  value, either increases or stays unchanged in each iteration. If the algorithm reaches the index  $i$ ,  $1 \leq i \leq \binom{n}{k}$ , such that the upper bound of  $\alpha_{k,K_i}$  for the  $i$ -th index set is already smaller than GLB, it will make GUB equal to GLB. At this point, GLB must be equal to  $\alpha_k$ , because the descending order of the upper bounds of  $\alpha_{k,K}$ , each index set  $K_j$  with  $j > i$  must have an  $\alpha_{k,K_j}$  that is smaller than GLB. In the meanwhile, as specified by SWA, GLB is the largest among  $\alpha_{k,K_j}$  with  $1 \leq j \leq (i-1)$ . So at this point, GLB must be the largest among  $\alpha_{k,K_j}$  with  $1 \leq j \leq \binom{n}{k}$ , namely  $\text{GLB} = \alpha_k$ .

If we cannot find such an index  $i$ , the algorithm will end up calculating  $\alpha_{k,K}$  for every index set  $K$  in the list. In this case, the upper and lower bounds will also become equal to  $\alpha_k$ , after each  $\alpha_{k,K}$  has been calculated.  $\square$

#### 4.2 Computational complexity

When  $l$  is fixed, the first step of SWA can be executed with polynomial-time complexity. The complexity of the second step is polynomial in  $n$ , if  $k$  is fixed. Furthermore, computing the upper bounds based on the pick- $l$ -element algorithm, and sorting index sets in order of the upper bounds are cheap in computation since computing the upper bounds requires only additive operations. Especially, when  $n$  and  $k$  are not big (e.g.  $n = 40$  and  $k = 5$ ), this second step can also be completed reasonably fast. The major complexity is due to the third step, which depends heavily on the number of index sets  $K$ , for which the algorithm will exactly compute  $\alpha_{k,K}$  until the upper and lower bounds coincide. This, in turn, depends on how tight the upper and lower bounds are during the execution of the algorithm.

In the worst case, the upper and lower bounds can meet after the  $\binom{n}{k}$  index sets  $K$  have been examined. However, in practice, we find that, very often, the upper and lower bounds meet very quickly, way before the algorithm examines  $\binom{n}{k}$  index sets. Thus, the algorithm will yield the exact value of  $\alpha_k$  with

---

**Algorithm 2** Sandwiching Algorithm

---

**Input:**  $L_j$  with  $|L_j| = l$  and  $\alpha_{l,L_j}$ ,  $j = 1, \dots, \binom{n}{l}$   
**Result:**  $\alpha_k$

- 1: **Initialize:**  $\text{GLB} \leftarrow 0$   $\triangleright$  GLB: Global Lower Bound
- 2:  $\text{CUB } \alpha_{k,K_j} \leftarrow$  solution of (2.3),  $\forall K_j, |K| = k, j = 1, \dots, \binom{n}{k}$   $\triangleright$  CUB: Cheap Upper Bound
- 3:  $Q \leftarrow$  sort  $\{K_j, j = 1, \dots, \binom{n}{k}\}$  in descending order of CUB
- 4: **for**  $i = 1$  to  $\binom{n}{k}$  **do**
- 5:   **if**  $\text{GLB} < \text{CUB } \alpha_{k,Q^{(i)}}$  **then**
- 6:      $\text{GUB} \leftarrow \text{CUB } \alpha_{k,Q^{(i)}}$   $\triangleright$  GUB: Global Upper Bound
- 7:      $\text{LPUB} \leftarrow$  solution of (4.1) for  $Q^{(i)}$   $\triangleright$  LPUB: Linear Programming Upper Bound
- 8:     **if**  $\text{GLB} < \text{LPUB}$  **then**
- 9:        $\alpha_{k,Q^{(i)}} \leftarrow$  solution of (2.1) for  $Q^{(i)}$
- 10:       **if**  $\alpha_{k,Q^{(i)}} > \text{GLB}$  **then**
- 11:           $\text{GLB} \leftarrow \alpha_{k,Q^{(i)}}$
- 12:       **end if**
- 13:     **end if**
- 14:   **else**
- 15:      $\text{GUB} \leftarrow \text{GLB}$
- 16:     **break**
- 17:   **end if**
- 18: **end for**
- 19:  $\alpha_k \leftarrow \text{GUB}$
- 20: **if**  $\alpha_k < \frac{1}{2}$  **then**
- 21:   NSC is satisfied
- 22: **else**
- 23:   NSC is not satisfied
- 24: **end if**

---

much lower computational complexity than the exhaustive search method. We provide an illustration of SWA in Fig. 1, and statistical performance in Fig. 4.

## 5. Tree Search Algorithm

The SWA requires calculation of upper bounds of  $\alpha_{k,K}$  for all the  $\binom{n}{k}$  index sets  $K$ ,  $|K| = k$ . This computation would take at least  $\binom{n}{k}$  memory units, and therefore, it becomes increasingly difficult to exactly verify the null space condition for high-dimensional sensing matrices. In order to mitigate these memory issues, we propose *Tree Search Algorithm (TSA)*, wherein representation of all the index sets in a tree structure greatly simplifies the search for maximum  $\alpha_{k,K}$  (or  $\alpha_k$  in short). Since a tree arranges its member sets based on some common properties, it avoids the large storage overhead common to all the index sets, thereby providing performance benefits.

TSA finds the optimal index set  $K^*$  which leads to the maximum value of  $\alpha_{k,K}$ 's over all possible index set  $K$ 's, i.e.,  $\alpha_{k,K^*} = \alpha_k$ , by applying a best-first branch-and-bound search strategy over the tree structure. In the tree structure, nodes represent signed subsets of  $\{1, \dots, n\}$  with an associated upper bound of  $\alpha_k$ . During the course of TSA execution, if this upper bound happens to be smaller than the previously computed lower bound of  $\alpha_k$ , the associated node and all its sub-nodes can be pruned. This approach is analogous to that of sequential decoding in error correcting codes [31], sphere decoder in

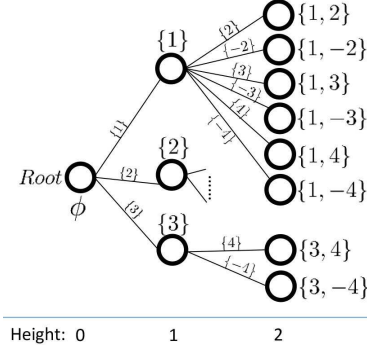


FIG. 2: Fully expanded tree structure following the legitimate order for  $k = 2$  and  $n = 4$ .

communication signal detection [35] and feature selection in pattern recognition [36].

Since TSA does not necessarily have to include every  $k$ -subset in the search process, TSA is faster than both the exhaustive search method and SWA, and requires less storage than SWA. We, however, note that SWA is relatively simpler to implement and more suitable for parallel computing. For the sake of simplicity, but without loss of generality, we introduce the 1-Step TSA, which is based on the pick-1-element algorithm. The  $l$ -Step TSA, then, is a simple extension of our techniques.

### 5.1 Tree structure and nodes

Throughout this section, we list several rules for constructing and expanding the tree for our algorithm.

**5.1.1 Tree structure.** A node  $J$  in the tree represents a signed index subset of  $\{1, \dots, n\}$  such that  $|J| \leq k$ . An edge in the tree represents an *edge subset* which has only one signed index. The index set that a node  $J$  represents is the union of all edge subsets that one encounters while traversing from the root node to the node  $J$ . Thus, we have the following rule:

**[R1]** “Parent node” - The parent node is a subset of its child node(s).

A node that has no children is referred to as a *leaf node*. We define the edge subset connecting the node  $J$  and its parent as the *incremental subset* of  $J$ . We call the cardinality of the index set that the node  $J$  represents as its *height*. A tree is considered *fully expanded* if every leaf node has cardinality  $k$ . In order to ensure distinct tree nodes represent different subsets, we have the following construction rule.

**[R2]** “Legitimate order” - Let  $A$  and  $B$  denote the incremental subset of a node  $J$ , and the incremental subset of  $J$ ’s child node respectively. Then, the absolute value of any index in  $B$  must be greater than that of any index in  $A$ .

Fig. 2 illustrates this rule for a fully-expanded tree with  $k = 2$  and  $n = 4$ . Note that the  $\alpha$  values have a sign symmetry with respect to the first and second indices e.g.,  $\alpha_{2,\{-1, 2\}} = \alpha_{2,\{1, -2\}}$ . Therefore, it suffices to consider the first index to be positive.

**5.1.2 Properties of a node.** As mentioned earlier, each node of the tree has an associated value of upper bound of  $\alpha_k$  to provide basis for further expansion and pruning of sub-nodes. It is important to

Table 1: Properties of a tree node  $J$ 

Property	Description
$B_1$	Upper bound of $\alpha_k$ via (5.1), where $B_{1d} = \min(B_2, B_3, B_4)$ , and $B_{1u}$ via (5.2) (If $j = k$ and the exact $\alpha_{j,J}$ is computed, it becomes a lower bound of $\alpha_k$ .)
$B_2$	Simple upper bound of $\alpha_{j,J}$ via (5.3)
$B_3$	Cheap upper bound of $\alpha_{j,J}$ via (2.3)
$B_4$	Exact $\alpha_{j,J}$ via (2.9)

obtain tighter upper bounds here so that more sub-nodes can be pruned thereby creating more storage space. However, computation of such tighter bounds increases the operational time. The TSA gets around this problem by successively computing inexpensive upper bounds to obtain a tighter bound at every node, as further explained below.

Let us consider a node  $J = \{J^{(1)}, J^{(2)}, \dots, J^{(j)}\}$ , such that  $|J| = j$ , and further assume that the node  $J$  could be a parent to some node  $K$ ,  $|K| = k$ , so that we have  $J \subseteq K$  (see [R1]). Then, we label the set  $\{J^{(1)}, J^{(2)}, \dots, J^{(j)}\}$  as the determined part of node  $J$  with maximum absolute index  $I_{\max}(J) = |J^{(j)}|$ . Next, we refer the set  $T = K \setminus J$  with  $|T| = t = k - j$  as the undetermined part that could result from a possible expansion from node  $J$ . The upper bound of  $\alpha_k$  at the node  $J$  is computed by accounting for both determined and undetermined parts:

$$B_1 = \underbrace{\text{upper bound of (or exact) } \alpha_{j,J}}_{B_{1d}} + \underbrace{\text{upper bound of } \alpha_{t,T}}_{B_{1u}}. \quad (5.1)$$

Here,  $B_{1u}$  is obtained by simply summing up the first  $t$  numbers of  $\alpha_{1,\{i_a\}}$ 's in descending order (see [R2]):

$$B_{1u} = \sum_{a=1}^t \alpha_{1,\{i_a\}}, \quad i_a \in \{1, \dots, n\}, \quad |i_a| \geq I_{\max}(J) + a + 1. \quad (5.2)$$

The  $B_{1d}$  is the minimum of three different upper bounds of  $\alpha_{j,J}$  that are obtained inexpensively at node  $J$ :  $B_2$ ,  $B_3$ , and  $B_4$ . The  $B_2$  is a simple upper bound of  $\alpha_{j,J}$  computed from the parent node as follows:

$$B_2 = \alpha_{1,\{J^{(j)}\}} + \text{upper bound of } \alpha_{j-1,\{J^{(1)}, J^{(2)}, \dots, J^{(j-1)}\}}, \quad (5.3)$$

where  $\{J^{(1)}, J^{(2)}, \dots, J^{(j-1)}\}$  represents the parent index set. We use the simple upper bound to choose the next node to appear in the searching tree. The bound  $B_3$  is the cheap upper bound of  $\alpha_{j,J}$  that we defined earlier in Lemma 2.1. The bound  $B_4$  is the exact  $\alpha_{j,J}$  itself. Note that for  $j = k$ , whenever TSA has computed the exact  $\alpha_{j,J}$  at the node,  $B_1$  of the node  $J$  becomes a lower bound of  $\alpha_k$ . Table 1 summarizes these properties.

The first or root node of TSA is always the empty set  $\emptyset$ . Until  $K^*$  is identified, TSA keeps expanding the tree as per the following rule:

**[R3]** “Expanding the tree from a node  $J$ ” - We first choose an edge subset  $\{i_a\}$  with the largest  $\alpha_{1,\{i_a\}}$  among unattached edge subsets that satisfy [R2]. The new node is then  $J' = J \cup \{i_a\}$  with following presets:  $B_2$  according to (5.3),  $B_3 = \infty$ ,  $B_4 = \infty$ ,  $B_{1d} = \min(B_2, B_3, B_4)$ , and  $B_{1u}$  according to (5.2).

## 5.2 Algorithm Description

The goal of TSA is to find a leaf node  $K$  that has the largest  $\alpha_{k,K}$  among all the nodes with cardinality  $k$ . The search strategy to achieve this goal is the branch-and-bound method [36]: for a node  $J$ , if its  $B_1$  (= an upper bound of  $\alpha_k$  at the node  $J$ ) is found to be smaller than the current lower bound on  $\alpha_k$ , then the child nodes of  $J$  are pruned. In the following, we explain the algorithm listing its important steps as *TSA00*, *TSA01*, and so on.

TSA consists of two steps: pre-computation [*TSA00*] and tree expansion [*TSA01*]. The pre-computation refers to computing  $\alpha_{1,\{i\}}$ ,  $i = 1, 2, \dots, n$ , and sorting the columns of  $A$  so that  $\alpha_{1,\{1\}} \geq \alpha_{1,\{2\}} \geq \dots \geq \alpha_{1,\{n\}}$ . The tree expansion is attaching new nodes to the tree starting with the *root node*, until every leaf node on the tree has a smaller  $B_1$  than  $\alpha_k$ . In the very beginning, the tree consists of only the root node. In each successive iteration, the TSA selects a leaf node  $J$  with the largest  $B_1$ . Here, the bound  $B_1$  of node  $J$  is also the global upper bound of  $\alpha_k$ . After the node  $J$  is found, TSA could either tighten the bound  $B_1$  of node  $J$ , or expand the tree from  $J$  to include more tree nodes (see [R3]). This process is repeated until *the selected node  $J$  is a  $k$ -element set and its bound  $B_1 = \alpha_{k,J}$* . Since the bound  $B_1$  of any other leaf node is not larger than  $\alpha_{k,J}$ , we have  $\alpha_{k,J} = \alpha_k$  and  $J = K^*$ .

Algorithm 3 shows a detailed pseudo-code of TSA where, for the sake of simplicity but without loss of generality,  $l = 1$ . As described in the pseudo-code, TSA creates a *priority queue* to store and sort all the leaf nodes in the current tree structure. The leaf nodes in the priority queue are called the *Candidate Nodes (CN's)* for expansion, and are sorted in the descending order of their  $B_1$  values. The node with the largest  $B_1$  in this priority queue is called the *Best Candidate Node (BCN)*. Once the BCN is identified, further operations depend on whether the  $B_3$  (i.e., the cheap upper bound) of the BCN has been computed [*TSA02*] or not [*TSA03*].

In case of *TSA02*, i.e.  $B_3 \neq \infty$ , TSA examines the computational status of its  $B_4$  value. If the  $B_4$  has been computed before [*TSA04*], then TSA expands the tree from the BCN by attaching the BCN's best direct child node (see [R3]) as a new CN. Thereafter, within the priority queue, TSA deletes the BCN and adds the new CN. If  $B_4$  already equals the exact  $\alpha_k$  and  $j = k$  [*TSA05*], TSA will terminate the search at the given BCN. Finally, if  $B_4 = \infty$  [*TSA06*], TSA will calculate the  $B_4$  value (and update  $B_1$ ) of the BCN. Irrespective of whether  $B_4 = \infty$  or not, TSA re-sorts the priority queue after these operations.

In case of *TSA03*, i.e.  $B_3 = \infty$ , the TSA will calculate the  $B_3$  value (and update  $B_1$ ). TSA then expands the tree from the parent node of this BCN by attaching a new CN (see [R3]). The priority queue adds the new CN and is re-sorted.

The search performance of the above-mentioned procedure is guaranteed by the following theorem.

**THEOREM 5.1** TSA provides the exact  $\alpha_k$  value.

We refer the reader to Appendix A.1 for the proof, and illustrate TSA for a simple example in Fig. 3.

## 6. Numerical Experiments

We conducted extensive simulations to compute the exact  $\alpha_k$  or its upper bounds using the pick- $l$ -element algorithm, SWA and TSA. For the same matrices, we compared our results with the previously mentioned approaches that use linear programming (LP) relaxation [16] and semidefinite programming (SDP) [9]. We assess the computational complexity in terms of execution time of the algorithm. We conducted our experiments on HP Z220 CMT with Intel Core i7-3770 dual core CPU @3.4GHz clock speed and 16GB DDR3 RAM, using Matlab (R2013b) on Windows 7 OS.

For small dimensional sensing matrices in Tables A.5 to A.8, we used CVX [14] - a package for

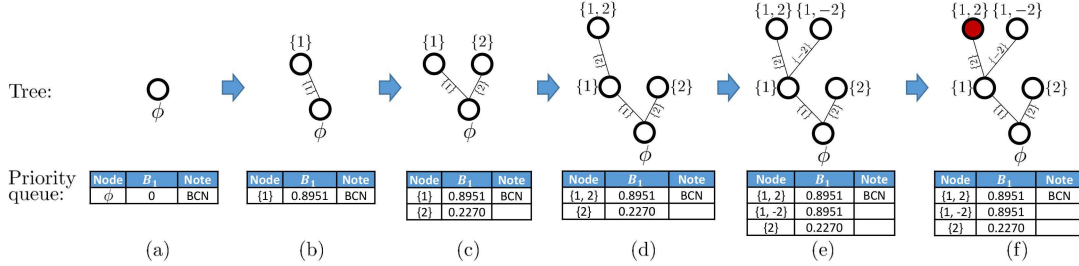


FIG. 3: An illustration of tree expansion step of TSA for  $(k, l) = (2, 1)$ , where  $\alpha_{1,\{1\}} = 0.7816$ ,  $\alpha_{1,\{2\}} = 0.1135$ , and  $\alpha_{1,\{3\}} = 0.1049$ . The figure shows the state of the priority queue *after* the each expansion of the tree. The red node in the last stage is the desired leaf node  $K^*$ . (a) Initialization. (b) TSA attaches the node  $\{1\}$  to the root node, because  $\alpha_{1,\{1\}}$  has the largest value among all  $\alpha_{1,\{i\}}$ 's. (c) TSA updates  $B_1$  as 0.8951 after computing the cheap upper bound of  $\alpha_{1,\{1\}}$ , and expands the tree by attaching the edge subset  $\{2\}$  to the parent node of  $\{1\}$ . (d) TSA expands the tree by attaching the node  $\{1, 2\}$  to the node  $\{1\}$ , and removes the node  $\{1\}$  from the priority queue, because  $\alpha_{1,\{1\}}$  has already been computed. (e) TSA calculates the cheap upper bound of  $\alpha_{2,\{1,2\}}$ , and adds  $\{1, -2\}$  to the parent of the BCN as a new CN. (f) TSA calculates the exact  $\alpha_{2,\{1,2\}} = 0.8951$ , namely, a lower bound of  $\alpha_2$ . TSA chooses the node  $\{1, 2\}$  as BCN again, and outputs  $\{1, 2\}$  as the optimal index set  $K^*$ , and  $\alpha_{2,\{1,2\}} = 0.8951$  as  $\alpha_2$ . Note that the  $B_1$  of  $\{1, -2\}$  is an upper bound of  $\alpha_2$ .

specifying and solving convex programs - to solve (2.1) and (2.9), and compared our algorithms with the codes for the SDP method from [9] which also uses CVX. For other numerical experiments, we used the commercial LP solver MOSEK [23] as in [16], to solve (2.9).

## 6.1 Low-dimensional sensing matrices

**6.1.1 Sensing matrices with  $n = 40$ .** We consider sensing matrices of row dimension  $m = 0.5n, 0.6n, 0.7n, 0.8n$ , where  $n = 40$ . We choose  $n = 40$  so that our results can be compared with the existing simulation results in [9]. For every matrix size, ten different realizations each of Gaussian and Fourier matrices were considered. So, we used total 80 different  $n = 40$  sensing matrices for the numerical experiments in Tables A.5 and A.8. We normalized all of the matrix columns so that they have a unit  $\ell_2$ -norm. The entries of Gaussian matrices were i.i.d standard Gaussian  $\mathcal{N}(0, 1)$ . The Fourier matrices had  $m$  of the rows as the Fourier bases drawn at random. To be consistent with the previous research, the Matlab code [16] provided at <http://www2.isye.gatech.edu/~nemirovs/> and the code provided by the authors of [9] are used for generating matrices and running LP and SDP methods. For readability, we place the numerical results for small sensing matrices with  $n = 40$  in Appendix A.2.

For a matrix of given size and type,  $k$  was increased from 1 to 5 in unit steps. Tables A.5 and A.7 show the *median* value of  $\alpha_k$ , and the obtained  $k_{max}$ , over all ten realizations. Compared to the previous work involving LP and SDP methods, we obtain more accurate  $\alpha_k$  values.

Tables A.6 and A.8 list the execution time *averaged* over 10 matrix realizations. Given our computational resources, we also estimated execution time (in minutes) to find the exact  $\alpha_k$  using the Exhaustive Search Method (ESM). Consider, for example, ESM takes 16 days to find  $\alpha_5$  for one instance of Gaussian or Fourier matrix given  $n = 40$ . We can contrast this with the execution time of the order of a few minutes that our algorithms take to execute. Note that LP and SDP methods do not provide an exact  $\alpha_k$  for  $k > 1$ .

We also provide simulation results demonstrating the distribution of execution time of SWA and

---

**Algorithm 3** Tree Search Algorithm based on the pick- $l$ -element algorithm ( $l$ -Step TSA)

---

**Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $k, l \in \mathbb{Z}^+$ **Result:**  $\alpha_k$ 

▷ **Pre-computation:** ▷ [TSA00]

- 1:  $\alpha_{1,\{i\}} \leftarrow$  solution of (2.1) for  $i = 1, \dots, n$
- 2: permute columns of  $A$  in descending order of  $\alpha_{1,\{i\}}$ 's
- 3:  $\alpha_{l,L_i} \leftarrow$  solution of (2.1),  $\forall L_i \subset \{1, \dots, n\}, |L_i| = l, i = 1, \dots, \binom{n}{l}$
- 4:  $\mathcal{Q} \leftarrow$  sort  $\{L_i, i = 1, \dots, \binom{n}{l}\}$  in descending order of  $\alpha_{l,L_i}$ 's like (2.5)

▷ **Tree expansion:** ▷ [TSA01]

- 5:  $B_i(\emptyset) \leftarrow 0, i = 1, 2, 3, 4$  ▷ Initialization
- 6: insert root node  $\emptyset$  into priority queue  $\Xi$
- 7: insert root node  $\emptyset$  into tree structure  $\mathcal{Y}$
- 8: **loop**
- 9:  $\text{BCN} \leftarrow \Xi(1)$  ▷  $\Xi(1)$  is the first node in  $\Xi$
- 10:  $h \leftarrow$  cardinality of BCN ▷ height of BCN
- 11: **if**  $B_3(\text{BCN}) = \infty$  **then** ▷ [TSA03]
- 12:  $B_3(\text{BCN}) \leftarrow$  cheap upper bound via (2.3)
- 13:  $B_1(\text{BCN}) \leftarrow B_{1d}(\text{BCN}) + B_{1u}(\text{BCN})$  ▷ See Table 1
- 14: expand  $\mathcal{Y}$  from the parent of BCN by using  $\mathcal{Q}$  ▷ See [R3]
- 15: insert the newly generated tree node  $J'$  to  $\Xi$
- 16: **else** ▷ [TSA02]
- 17: **if**  $B_4(\text{BCN}) = \infty$  **then** ▷ [TSA06]
- 18:  $B_4(\text{BCN}) \leftarrow$  exact  $\alpha_{k,\text{BCN}}$  via (2.9)
- 19:  $B_1(\text{BCN}) \leftarrow B_{1d}(\text{BCN}) + B_{1u}(\text{BCN})$  ▷ See Table 1
- 20: **else**
- 21: **if**  $h = k$  **then** ▷ [TSA05]
- 22:  $\alpha_k \leftarrow B_4(\text{BCN})$
- 23: **break**
- 24: **else** ▷ [TSA04]
- 25: expand  $\mathcal{Y}$  from BCN by using  $\mathcal{Q}$  ▷ See [R3]
- 26: insert the newly generated tree node  $J'$  to  $\Xi$
- 27: remove BCN from  $\Xi$
- 28: **end if**
- 29: **end if**
- 30: **end if**
- 31: sort  $\Xi$  in descending order of  $B_1$
- 32: **end loop**

---

TSA. For  $m = 0.5n$ , we generated 100 random realizations of Gaussian matrices and computed  $\alpha_5$  using SWA and TSA (based on the pick-3-element algorithm). Fig.4 shows the distribution of the execution time and number of steps in SWA or number of nodes located in height 5 in TSA. The maximum number of steps that SWA may take is  $\binom{n}{k} = \binom{40}{5} = 6.58008e5$ . From Fig.4 (b), we note that, for 90% of the cases, SWA completes the execution in about  $6e3$  steps out of  $6.58e5$  worst-case steps, which is 1% of the worst-case steps. For 90 % of the simulated matrices, TSA terminates its execution time before 1.6 % of height-5 nodes ( $1.7e5$  nodes out of  $1.05e7$  total possible height-5 nodes) are attached to the



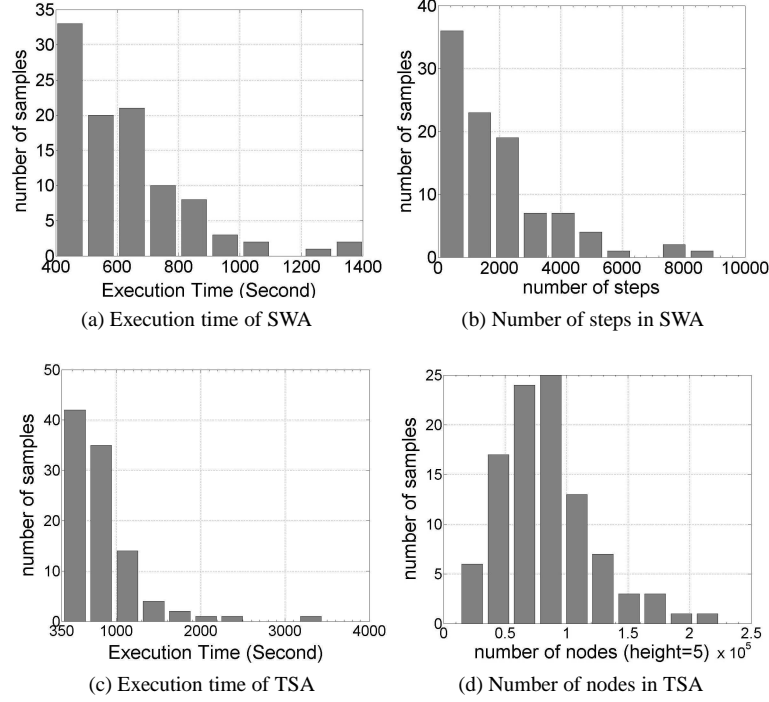
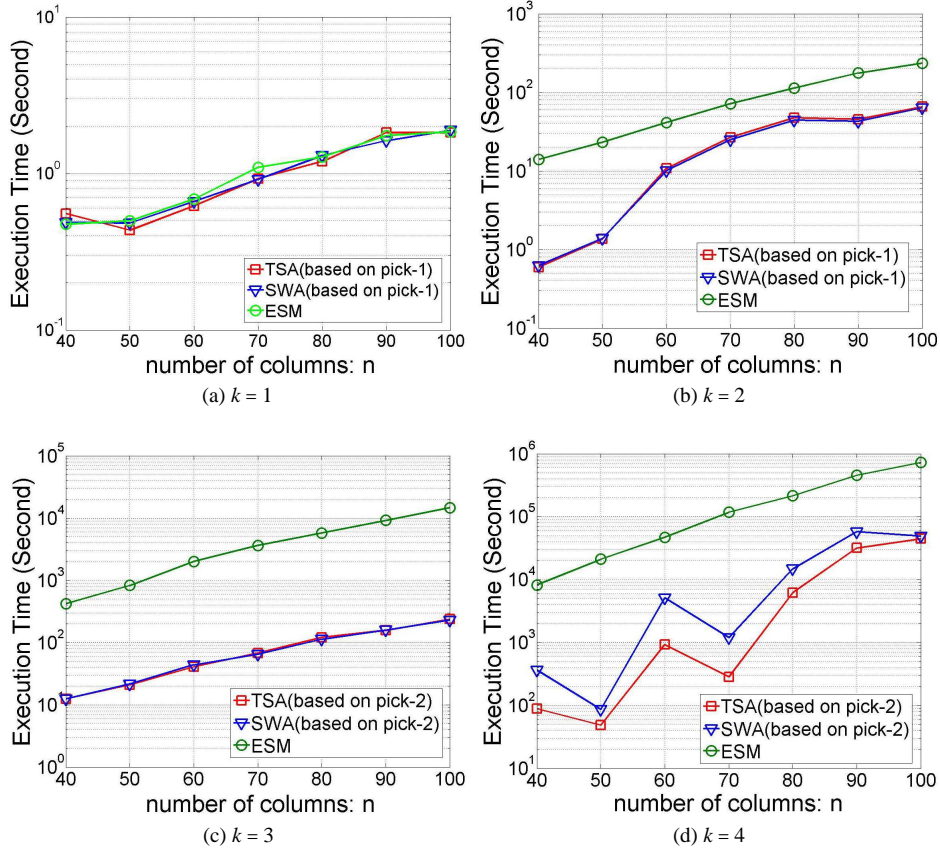
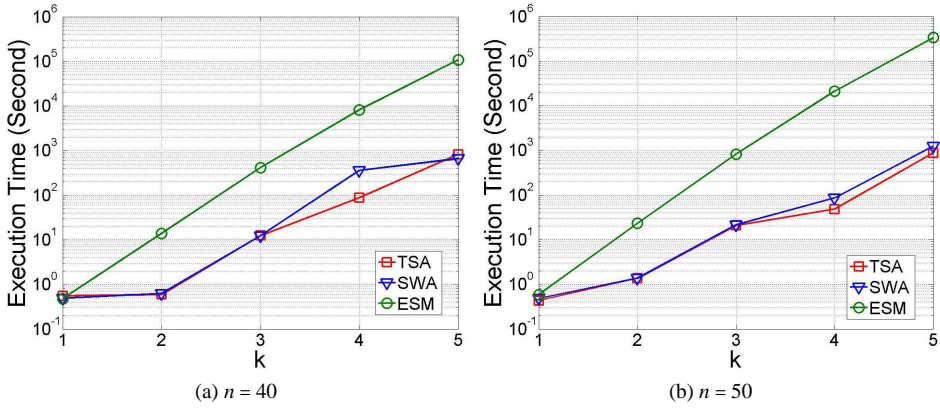


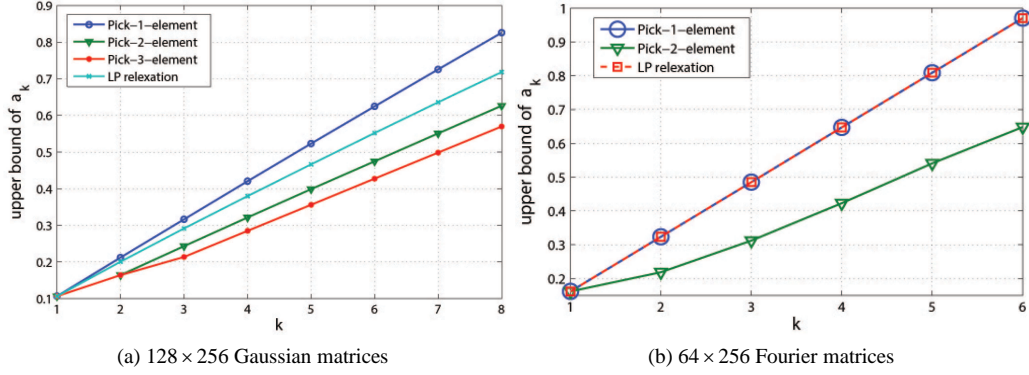
FIG. 4: Histograms of SWA and TSA (based on the pick-3) to find  $\alpha_5$  on 100 randomly chosen  $20 \times 40$  Gaussian sensing matrices for each method.

tree. We remark that the number of height-5 nodes indicates the subsets for which we will calculate the exact  $\alpha_{k,K}$ .

In addition, we provide the execution time of SWA and TSA for different  $n$  with fixed  $k$  in Fig. 5, and for different  $k$  with fixed  $n$  in Fig. 6. The execution time of ESM is calculated by multiplying the average time to solve (2.9) for an index set by the total number of index sets.

**6.1.2 Sensing matrices with  $n = 256$ .** We assessed the performance of the pick- $l$ -element algorithm for sensing matrices with  $n = 256$ . We carried out numerical experiments on  $128 \times 256$  Gaussian matrices in Fig. 7 (a) and  $64 \times 256$  Fourier matrices in Fig. 7 (b). Here, for 10 sensing matrices, we obtained the median values of the upper bounds of  $\alpha_k$  using the pick- $l$ -element algorithms and compared the results with the LP relaxation method [16]. We omit the SDP method [9] from this experiment due to its very high computational complexity for  $n = 256$ . For the pick-3-element algorithm in Fig. 7 (a), we calculated the upper bound of  $\alpha_3$  via TSA, and used this result to calculate upper bound of  $\alpha_k$ ,  $k = 3, 4, \dots, 8$  via (2.7). Fig. 7 (a) and 7 (b) demonstrate that, with an appropriate choice of  $l$ , the upper bound of  $\alpha_k$  obtained via the pick- $l$ -element can be tighter than that from the LP relaxation method. For example, for  $64 \times 256$  Fourier matrices, the improvement of pick-2-element algorithm over LP relaxation is quite significant. In fact, for  $64 \times 256$  Gaussian matrices, LP relaxation often determines the maximum recoverable sparsity as 2, while the pick-2-element algorithm improves it to 4.

FIG. 5: The execution time of SWA and TSA in log scale as a function of  $n$  ( $m = n/2$ ).FIG. 6: The execution time of SWA and TSA in log scale as a function of  $k$  ( $m = n/2$ ).

FIG. 7: Median upper bounds of  $\alpha_k$  from the pick- $l$ -element algorithm and the LP relaxation method.Table 2: Lower bound on  $k$  and execution time (Gaussian Matrix with  $n = 1024$ )

(a) Lower bound on $k$						(b) Execution time (Unit: second)					
matrix $A$	Pick-1	(2.6) <sup>d</sup>	$0.5/\alpha_1^e$	$k(\alpha_1)^a$	SDP <sup>b</sup>	matrix $A$	Pick-1	(2.6)	$0.5/\alpha_1$	$k(\alpha_1)^a$	SDP <sup>b</sup>
$102 \times 1024$	2	3	2	2	N/A <sup>c</sup>	$102 \times 1024$	237	1 day <sup>f</sup>	237	200	N/A <sup>c</sup>
$205 \times 1024$	4	4	4	4	N/A	$205 \times 1024$	452	1 day	452	429	N/A
$307 \times 1024$	5	6	5	5	N/A	$307 \times 1024$	796	1 day	796	723	N/A
$410 \times 1024$	7	8	7	7	N/A	$410 \times 1024$	1207	1 day	1207	1073	N/A
$512 \times 1024$	9	10	9	9	N/A	$512 \times 1024$	1952	1 day	1952	1600	N/A
$614 \times 1024$	12	13	12	12	N/A	$614 \times 1024$	2150	1 day	2150	2217	N/A
$717 \times 1024$	16	17	15	16	N/A	$717 \times 1024$	1337	1 day	1337	2992	N/A
$819 \times 1024$	21	23	20	21	N/A	$819 \times 1024$	838	1 day	838	3904	N/A
$922 \times 1024$	32	36	30	32	N/A	$922 \times 1024$	386	1 day	386	4730	N/A

<sup>a</sup> Linear Programming [16]<sup>b</sup> Semidefinite Programming [9]<sup>c</sup> Not Available<sup>d</sup> Lower bound on  $k$  by using the equation (2.6) with upper bound on  $\alpha_2$ <sup>e</sup> Lower bound on  $k$  according to [16, Section 4.2.B] with  $\alpha_1$  obtained from our own pick-1-element algorithm.<sup>f</sup> Upper bound on  $\alpha_2$  obtained from 1-Step TSA after 1 day's run

## 6.2 High-dimensional sensing matrices

We further conducted numerical experiments for even higher dimensional Gaussian sensing matrices from  $n = 1024$  to  $n = 6144$ . For these numerical experiments in Tables 2a to 3b, we calculated the lower bound on recoverable  $k$  and obtained the execution time of the pick- $l$ -element algorithm. The SDP method [9] was not available to these experiments due to its very high computational complexity. For  $\alpha_1$  in Tables 2a to 3b, we ran our pick-1-element algorithm, and calculated the lower bound on  $k$  from  $0.5/\alpha_1$  [16, Section 4.2.B]. For  $\alpha_2$  in Table 2a and 2b, we ran TSA for 1 day (24 hours), and obtained an upper bound on  $\alpha_2$ , which is  $B_1$  of the Best Candidate Node at the time of algorithm termination (this property is provided in the proof of the optimality of TSA). With the upper bound on  $\alpha_2$ , we obtained the lower bound on  $k$  via (2.6) in Lemma 2.3.

We note that, the LP based method in [16] obtains the same  $\alpha_1$  as our pick-1-element algorithm, but the complexity of the LP method (namely, the equation (4.29) in [16]) to calculate  $\alpha_1$  is very high for higher dimensional matrices. Moreover, we observe that the recoverable sparsity bound  $0.5/\alpha_1$

Table 3: Lower bound on  $k$  and execution time (Gaussian Matrix)

(a) Lower bound on $k$					(b) Execution time (Unit: second)				
matrix $A$	Pick-1	$0.5/\alpha_1^d$	$k(\alpha_1)^a$	SDP <sup>b</sup>	matrix $A$	Pick-1	$0.5/\alpha_1$	$k(\alpha_1)$	SDP
$2007 \times 2048$	102	90	102	N/A <sup>c</sup>	$2007 \times 2048$	671	671	71948	N/A
$4014 \times 4096$	152	139	N/A	N/A	$4014 \times 4096$	9116	9116	15 days <sup>e</sup>	N/A
$6021 \times 6144$	190	174	N/A	N/A	$6021 \times 6144$	38935	38935	65.5 days <sup>f</sup>	N/A
$6134 \times 6144$	558	406	N/A	N/A	$6134 \times 6144$	13734	13734	41.7 days <sup>g</sup>	N/A

<sup>a</sup> Linear Programming[16]<sup>b</sup> Semidefinite Programming[9]<sup>c</sup> Not Available<sup>d</sup> Lower bound on  $k$  according to [16, Section 4.2.B] with  $\alpha_1$  obtained from our own pick-1-element algorithm.<sup>e</sup> Estimated time (15 hours for finishing 4% calculations with the provided Matlab code [16])<sup>f</sup> Estimated time (15 hours for finishing 1% calculations with the provided Matlab code [16])<sup>g</sup> Estimated time (10 hours for finishing 1% calculations with the provided Matlab code [16])

is smaller than the recoverable sparsity bound provided by our pick-1-element algorithm. Although the improved recoverable sparsity bound  $k(\alpha_1)$  from [16] is similar to the recoverable sparsity bound provided by the pick-1-element algorithm, the speed to obtain  $k(\alpha_1)$  bound can be quite slow when the dimension of the sensing matrix is large.

Our numerical results in Tables 2a to 3b clearly show that our pick- $l$ -element algorithm outperforms LP and SDP based methods in recoverable sparsity  $k$  and the execution time. It is worth mentioning that even for extremely large sensing matrices, e.g.,  $4014 \times 4096$  and  $6021 \times 6144$ , where LP and SDP cannot provide any lower bound on  $k$  due to unreasonable computational time, our pick- $l$ -element algorithm can provide the lower bound on  $k$  efficiently. Tables 3a and 3b show the lower bound on  $k$  and the execution time in detail for these large dimensional matrices, where our verified recoverable sparsity  $k$  can be as large as 558.

### 6.3 Application to Network Tomography Problem

We apply our new tools introduced in this paper to verify sensing matrices in the network tomography problems [34, 12, 7, 28, 29, 6]. In an undirected complete graph model for the communication network, the communication delay over each link can be determined by sending packets through probing paths that are composed of connected links. The delay of each path is then measured by adding the delays over its links. Generally most links are uncongested, and only a few congested links have significant delays. It is, therefore, reasonable to think of finding the link delays as a sparse recovery problem. This sparse problem can be expressed in a system of linear equations  $y = Ax$ , where the vector  $y \in \mathbb{R}^m$  is the delay of  $m$  paths, the vector  $x \in \mathbb{R}^n$  is the delay vector for the  $n$  links, and  $A$  is the sensing matrix. The element  $A_{ij}$  of  $A$  is 1, if and only if path  $y_i$ ,  $i \in \{1, 2, \dots, m\}$ , goes through link  $j$ ,  $j \in \{1, 2, \dots, n\}$ ; otherwise  $A_{ij}$  equal to 0 (Fig. 8). The indices of nonzero elements in the vector  $x$  correspond to the congested links.

In our numerical experiments to verify the NSC in a network tomography problem, the paths for sending data packets were generated by random walks of fixed length. Tables 4a-4b summarize the results of our experiments. We note that using TSA one can *exactly* verify that a total of  $k = 3$  and  $k = 4$  congested link delays can be uniquely found by solving  $\ell_1$  minimization problem (1.2) for the randomly generated network measurement matrices  $33 \times 66$  (12-node complete graph) and  $53 \times 105$  (15-node complete graph) respectively. We note that the exhaustive search method would take about 2 years to exactly verify the NSC for a  $53 \times 105$  matrix.

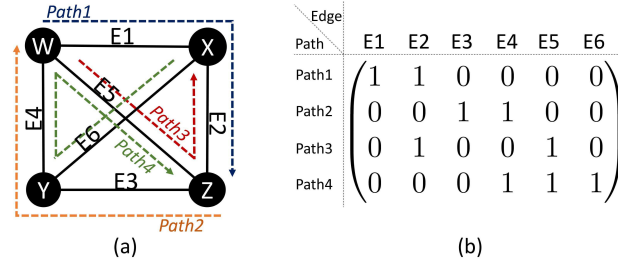


FIG. 8: (a) A network tomography graph.  $W$ ,  $X$ ,  $Y$ , and  $Z$  are nodes in the network, and  $Path1$ , 2, 3, and 4 are the probing paths through which the packets are sent. (b) The measurement or sensing matrix [34] corresponding to the graph shown in (a). The rows and columns of this matrix represent probing paths and edges respectively.

Table 4:  $\alpha_k$  and execution time in Network Tomography Problem

(a)  $\alpha_k$  values (Rounded off to the nearest hundredth)

matrix $A$	Algo.	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$	$k_{max}$
$33 \times 66^a$	TSA	0.22	0.35	0.45	0.53	-	3
$53 \times 105^b$	TSA	0.22	0.33	0.41	0.47	0.52	4

<sup>a</sup> Random walk step: 10

<sup>b</sup> Random walk step: 20

(b) Execution time (Unit: second)

matrix $A$	Algo.	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$
$33 \times 66$	TSA	4 <sup>a</sup>	12 <sup>a</sup>	44 <sup>b</sup>	525 <sup>b</sup>	-
	ESM <sup>d</sup>	4	92	3.9e3	1.2e5	-
$53 \times 105$	TSA	4 <sup>a</sup>	18 <sup>a</sup>	8.6e3 <sup>c</sup>	8.6e3 <sup>c</sup>	5.8e4 <sup>c</sup>
	ESM	4	432	3.0e4	1.5e6	6.1e7

<sup>a</sup> 1-Step TSA

<sup>b</sup> 2-Step TSA

<sup>c</sup> 3-Step TSA

<sup>d</sup> Exhaustive Search Method (Estimated Operation time = average time to solve (2.9) for an index set <sup>e</sup>  $\times$  total number of index sets)

<sup>e</sup> 0.02 sec ( $33 \times 66$  matrix), 0.04 sec ( $53 \times 105$  matrix) are obtained from 100 samples

## Acknowledgement

We thank Alexandre d'Aspremont from CNRS at Ecole Normale Suprieure in Paris, and Anatoli Juditsky from Georgia Institute of Technology for helpful discussions and providing codes for the simulations in [9] and [16].

## References

- [1] BOBIN, J., STARCK, J.-L. & OTTENSAMER, R. (2008) Compressed sensing in astronomy. *IEEE Journal of Selected Topics in Signal Processing*, **2**(5), 718–726.
- [2] BRADY, D. J., CHOI, K., MARKS, D. L., HORISAKI, R. & LIM, S. (2009) Compressive holography. *Optics express*, **17**(15), 13040–13049.
- [3] CANDÈS, E. J., ROMBERG, J. & TAO, T. (2006a) Robust uncertainty principles: Exact signal

- reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, **52**(2), 489–509.
- [4] ——— (2006b) Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, **59**(8), 1207–1223.
  - [5] CANDÈS, E. J. & TAO, T. (2005) Decoding by linear programming. *IEEE Transactions on Information Theory*, **51**(12), 4203–4215.
  - [6] CASTRO, R., COATES, M., LIANG, G., NOWAK, R. & YU, B. (2004) Network tomography: recent developments. *Statistical Science*, **19**(3), 499–517.
  - [7] COATES, M. J. & NOWAK, R. D. (2001) Network tomography for internal delay estimation. in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 6, pp. 3409–3412.
  - [8] COHEN, A., DAHMEN, W. & DEVORE, R. (2009) Compressed sensing and best  $k$ -term approximation. *Journal of the American Mathematical Society*, **22**(1), 211–231.
  - [9] D’ASPREMONT, A. & EL GHAOU, L. (2011) Testing the nullspace property using semidefinite programming. *Mathematical programming*, **127**(1), 123–144.
  - [10] DONOHO, D. (2005) Neighborly Polytopes and Sparse Solution of Underdetermined Linear Equations. *Technical report (Stanford University. Dept. of Statistics)*.
  - [11] ENDER, J. H. G. (2010) On compressive sensing applied to radar. *Signal Processing*, **90**(5), 1402–1414.
  - [12] FIROOZ, M. H. & ROY, S. (2010) Network tomography via compressed sensing. in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 1–5.
  - [13] GAMPER, U., BOESIGER, P. & KOZERKE, S. (2008) Compressed sensing in dynamic MRI. *Magnetic Resonance in Medicine*, **59**(2), 365–373.
  - [14] GRANT, M. & BOYD, S. (2012) CVX: Matlab Software for Disciplined Convex Programming, version 2.0 beta. <http://cvxr.com/cvx>.
  - [15] HARVEY, N. J., PATRASCU, M., WEN, Y., YEKHANIN, S. & CHAN, V. W. (2007) Non-Adaptive Fault Diagnosis for All-Optical Networks via Combinatorial Group Testing on Graphs. in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pp. 697–705.
  - [16] JUDITSKY, A. & NEMIROVSKI, A. (2011) On verifiable sufficient conditions for sparse signal recovery via  $\ell_1$  minimization. *Mathematical programming*, **127**(1), 57–88.
  - [17] JUNG, H., SUNG, K., NAYAK, K. S., KIM, E. Y. & YE, J. C. (2009) k-t FOCUSS: A general compressed sensing framework for high resolution dynamic MRI. *Magnetic Resonance in Medicine*, **61**(1), 103–116.
  - [18] LEE, K. & BRESLER, Y. (2008) Computing performance guarantees for compressed sensing. in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5129–5132.

- [19] LIU, Y., NING, P. & REITER, M. K. (2011) False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)*, **14**(1), 13.
- [20] LUSTIG, M., DONOHO, D. & PAULY, J. M. (2007) Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, **58**(6), 1182–1195.
- [21] LUSTIG, M., DONOHO, D., SANTOS, J. & PAULY, J. (2008) Compressed Sensing MRI. *IEEE Signal Processing Magazine*, **25**(2), 72–82.
- [22] MARIM, M. M., ATLAN, M., ANGELINI, E. & OLIVO-MARIN, J.-C. (2010) Compressed sensing with off-axis frequency-shifting holography. *Optics letters*, **35**(6), 871–873.
- [23] MOSEK APS (2015) *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 31)*.
- [24] NAGESH, P. & LI, B. (2009) A compressive sensing approach for expression-invariant face recognition. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1518–1525.
- [25] RIVENSON, Y., STERN, A. & JAVIDI, B. (2010) Compressive fresnel holography. *Journal of Display Technology*, **6**(10), 506–509.
- [26] TANG, G. & NEHORAI, A. (2011) Verifiable and Computable  $\ell_\infty$  Performance Evaluation of  $\ell_1$  Sparse Signal Recovery. in *Proceedings of 45th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6.
- [27] TILLMANN, A. M. & PFETSCH, M. E. (2014) The Computational Complexity of the Restricted Isometry Property, the Nullspace Property, and Related Concepts in Compressed Sensing. *IEEE Transactions on Information Theory*, **60**(2), 1248–1259.
- [28] TSANG, Y., COATES, M. & NOWAK, R. D. (2003) Network delay tomography. *IEEE Transactions on Signal Processing*, **51**(8), 2125–2136.
- [29] VARDI, Y. (1996) Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American Statistical Association*, **91**(433), 365–377.
- [30] WAGNER, A., WRIGHT, J., GANESH, A., ZHOU, Z., MOBAHI, H. & MA, Y. (2012) Toward a Practical Face Recognition System: Robust Alignment and Illumination by Sparse Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **34**(2), 372–386.
- [31] WOZENCRAFT, J. & REIFFEN, B. (1961) *Sequential Decoding*. Technology Press of the Massachusetts Institute of Technology and Wiley.
- [32] WRIGHT, J., YANG, A. Y., GANESH, A., SASTRY, S. S. & MA, Y. (2009) Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **31**(2), 210–227.
- [33] XU, W. & HASSIBI, B. (2011) Precise Stability Phase Transitions for  $\ell_1$  Minimization: A Unified Geometric Framework. *IEEE Transactions on Information Theory*, **57**(10), 6894–6919.

- [34] XU, W., MALLADA, E. & TANG, A. (2011) Compressive sensing over graphs. in *Proceedings of the 30th IEEE International Conference on Computer Communizations (INFOCOM)*, pp. 2087–2095.
- [35] XU, W., WANG, Y., ZHOU, Z. & WANG, J. (2004) A computationally efficient exact ML sphere decoder. in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 2594–2598.
- [36] YU, B. & YUAN, B. (1993) A more efficient branch and bound algorithm for feature selection. *Pattern Recognition*, **26**(6), 883–889.

## Appendix

### A.1 Proof of Theorem 5.1

*Proof.* TSA terminates its operation, only when the height of the BCN is  $k$  and the  $B_1$  of the BCN is  $\alpha_{k,K}$ , where  $K$  is the index set of the BCN. Since the algorithm picks  $K$  as the best candidate node, all the other nodes in the priority queue have no larger  $B_1$  value than the BCN  $K$ .

By the expansion of the tree in TSA, each subset  $K'$  with  $|K'| = k$  must either be a superset (a child) of a CN in the priority queue, or only of the direct parent of a CN in the priority queue. Let us assume that  $K'$  is a superset of a CN  $J$  appearing in the priority queue. Then the  $B_1$  of  $J$  is not larger than the  $B_1$  of the BCN (which is  $\alpha_{k,K}$ ), that is,  $B_1(J) \leq \alpha_{k,K}$ . At the same time, since the  $B_1(J)$  is an upper bound of  $\alpha_{k,K'}$  for all the subsets  $K'$  with  $J \subseteq K'$ , we have  $B_1(J) \geq \alpha_{k,K'}$ . From these two inequalities, we clearly have  $\alpha_{k,K'} \leq \alpha_{k,K}$ .

Suppose that  $K'$  with  $|K'| = k$  is not a superset of any CN, but instead a superset of the direct parent node  $J \setminus \{J^{(j)}\}$  of a CN  $J$  appearing in the priority queue. Then  $K'$  is a superset of another child node  $J'$  of  $J \setminus \{J^{(j)}\}$  where  $J'$  has not been attached to the tree in TSA. By the way of attaching a new CN to a tree,  $B_1(J') \leq B_1(J)$ . Because  $B_1(J) \leq \alpha_{k,K}$ , we have  $B_1(J') \leq \alpha_{k,K}$ . Since  $B_1(J')$  is an upper bound of  $\alpha_{k,K'}$  for all the subsets  $K'$  with  $J' \subseteq K'$ ,  $\alpha_{k,K'} \leq \alpha_{k,K}$ .

In summary, for any subset  $K'$  with  $|K'| = k$ ,  $\alpha_{k,K'} \leq \alpha_{k,K}$ . Therefore,  $\alpha_{k,K}$  is equal to  $\alpha_k$ .  $\square$

### A.2 Low-dimensional sensing matrices with $n = 40$

Here, we provide the numerical results for small sensing matrices with  $n = 40$  to compare our methods to LP [16] and SDP [9] methods. For these numerical experiments, we used CVX [14].



Table A.5:  $\alpha_k$  comparison (Gaussian Matrix) - Median from 10 samples

(Rounded off to the nearest hundredth)							
matrix $A$ ( $m \times n$ )	Algo.	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$	$k_{max}$ (median / avg. <sup>d</sup> )
$20 \times 40$ ( $\rho^a = 0.5$ )	pick-1	0.28	0.55	0.81	1.08	1.32	1 / 1.1
	pick-2	-	0.44	0.65	0.85	1.03	2 / 1.9
	pick-3	-	-	0.57	0.74	0.90	- / -
	TSA	0.28	0.44	0.57	0.66	0.74	2 / 1.9
	SWA	-	0.44	0.57	0.66	0.74	2 / 1.9
	LP <sup>b</sup>	0.28	0.49	0.67	0.83	0.97	2 / 1.6
	SDP <sup>c</sup>	0.28	0.48	0.65	0.81	0.94	2 / 1.6
$24 \times 40$ ( $\rho = 0.6$ )	pick-1	0.25	0.47	0.69	0.90	1.09	2 / 2.0
	pick-2	-	0.39	0.56	0.72	0.88	2 / 2.0
	pick-3	-	-	0.49	0.64	0.78	3 / 2.5
	TSA	0.25	0.39	0.49	0.58	0.65	3 / 2.5
	SWA	-	0.39	0.49	0.58	0.65	3 / 2.5
	LP	0.25	0.41	0.58	0.71	0.84	2 / 2.0
	SDP	0.25	0.41	0.56	0.70	0.83	2 / 2.0
$28 \times 40$ ( $\rho = 0.7$ )	pick-1	0.21	0.40	0.57	0.73	0.88	2 / 2.1
	pick-2	-	0.33	0.47	0.61	0.74	3 / 2.8
	pick-3	-	-	0.41	0.54	0.66	3 / 3.0
	TSA	0.21	0.33	0.41	0.49	0.57	4 / 3.6
	SWA	-	0.33	0.41	0.49	0.57	4 / 3.6
	LP	0.21	0.35	0.49	0.61	0.72	3 / 2.8
	SDP	0.21	0.35	0.48	0.60	0.71	3 / 2.9
$32 \times 40$ ( $\rho = 0.8$ )	pick-1	0.15	0.30	0.44	0.57	0.69	3 / 3.0
	pick-2	-	0.26	0.38	0.50	0.60	4 / 3.7
	pick-3	-	-	0.36	0.47	0.56	4 / 4.0
	TSA	0.15	0.26	0.36	0.44	0.50	4 / 4.5
	SWA	-	0.26	0.36	0.44	0.50	4 / 4.5
	LP	0.15	0.28	0.39	0.49	0.58	4 / 3.8
	SDP	0.15	0.28	0.39	0.49	0.58	4 / 3.9

<sup>a</sup>  $\rho = m/n$     <sup>b</sup> Linear Programming [16]    <sup>c</sup> Semidefinite Programming [9]

<sup>d</sup> Average of the largest recoverable sparsity from 10 samples

Table A.6: Operation time (Gaussian Matrix) - Average from 10 samples

Unit: minute (Rounded off to the nearest hundredth)						
matrix $A$ ( $m \times n$ )	Algo.	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$
$20 \times 40$ ( $\rho^a = 0.5$ )	pick-1	0.10	0.10	0.10	0.10	0.10
	pick-2	-	2.95	2.95	2.95	2.95
	pick-3	-	-	106.18	106.18	106.18
	TSA	0.11 <sup>d</sup>	0.83 <sup>d</sup>	4.50 <sup>e</sup>	13.47 <sup>e</sup>	89.94 <sup>f</sup>
	SWA	-	0.76 <sup>a</sup>	3.28 <sup>b</sup>	106.89 <sup>c</sup>	126.14 <sup>c</sup>
	LP <sup>g</sup>	0.01	0.01	0.01	0.01	0.01
	SDP <sup>h</sup>	0.86	79.66	68.77	68.02	76.85
	ESM <sup>i</sup>	0.09	3.34	84.60	1.565e3	2.2537e4
$24 \times 40$ ( $\rho = 0.6$ )	pick-1	0.07	0.07	0.07	0.07	0.07
	pick-2	-	4.30	4.30	4.30	4.30
	pick-3	-	-	105.55	105.55	105.55
	TSA	0.14 <sup>d</sup>	0.50 <sup>d</sup>	4.27 <sup>e</sup>	6.86 <sup>e</sup>	85.72 <sup>f</sup>
	SWA	-	0.43 <sup>a</sup>	3.09 <sup>b</sup>	105.88 <sup>c</sup>	113.92 <sup>c</sup>
	LP	0.01	0.01	0.01	0.01	0.01
	SDP	0.91	72.74	68.34	64.51	69.43
	ESM	0.08	3.25	82.31	1.522e3	2.1927e4
$28 \times 40$ ( $\rho = 0.7$ )	pick-1	0.11	0.11	0.11	0.11	0.11
	pick-2	-	4.28	4.28	4.28	4.28
	pick-3	-	-	106.16	106.16	106.16
	TSA	0.12 <sup>d</sup>	0.36 <sup>d</sup>	4.27 <sup>e</sup>	4.93 <sup>e</sup>	35.60 <sup>e</sup>
	SWA <sup>b</sup>	-	4.28	4.39	7.82	436.99
	LP	0.02	0.01	0.01	0.01	0.02
	SDP	0.88	83.10	70.82	65.48	65.82
	ESM	0.08	3.28	83.09	1.537e3	2.2136e4
$32 \times 40$ ( $\rho = 0.8$ )	pick-1	0.11	0.11	0.11	0.11	0.11
	pick-2	-	4.23	4.23	4.23	4.23
	pick-3	-	-	109.33	109.33	109.33
	TSA	0.12 <sup>d</sup>	0.24 <sup>d</sup>	4.39 <sup>e</sup>	4.51 <sup>e</sup>	6.09 <sup>e</sup>
	SWA <sup>b</sup>	-	4.23	4.28	4.78	17.58
	LP	0.01	0.01	0.01	0.01	0.05
	SDP	0.97	100.90	76.99	71.67	63.60
	ESM	0.09	3.33	84.38	1.561e3	2.2480e4

<sup>a</sup> Sandwiching Algorithm based on the pick-1-element algorithm<sup>b</sup> Sandwiching Algorithm based on the pick-2-element algorithm<sup>c</sup> Sandwiching Algorithm based on the pick-3-element algorithm<sup>d</sup> 1-Step TSA    <sup>e</sup> 2-Step TSA    <sup>f</sup> 3-Step TSA<sup>g</sup> Linear Programming [16]    <sup>h</sup> Semidefinite Programming [9]<sup>i</sup> Exhaustive Search Method (Estimated Operation time = average time to solve (2.9) for an index set from 100 samples  $\times$  total number of index sets)

Table A.7:  $\alpha_k$  comparison (Fourier Matrix) - Median from 10 samples

(Rounded off to the nearest hundredth)							
matrix $A$ ( $m \times n$ )	Algo.	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$	$k_{max}$ (median / avg. <sup>d</sup> )
$20 \times 40$ ( $\rho^a = 0.5$ )	pick-1	0.19	0.39	0.58	0.77	0.96	2 / 2.0
	pick-2	-	0.34	0.50	0.67	0.84	2 / 2.5
	pick-3	-	-	0.47	0.63	0.78	3 / 2.7
	TSA	0.19	0.34	0.47	0.60	0.71	3 / 2.7
	SWA	-	0.34	0.47	0.60	0.71	3 / 2.7
	LP <sup>b</sup>	0.19	0.39	0.58	0.77	0.96	2 / 2.0
	SDP <sup>c</sup>	0.19	0.39	0.58	0.77	0.96	2 / 2.0
$24 \times 40$ ( $\rho = 0.6$ )	pick-1	0.16	0.31	0.47	0.62	0.78	3 / 3.0
	pick-2	-	0.28	0.41	0.55	0.69	3 / 3.1
	pick-3	-	-	0.38	0.51	0.64	3 / 3.2
	TSA	0.16	0.28	0.38	0.50	0.63	3 / 3.3
	SWA	-	0.28	0.38	0.50	0.63	3 / 3.3
	LP	0.16	0.31	0.47	0.62	0.78	3 / 3.0
	SDP	0.16	0.31	0.47	0.62	0.78	3 / 3.0
$28 \times 40$ ( $\rho = 0.7$ )	pick-1	0.13	0.26	0.40	0.53	0.66	3 / 3.3
	pick-2	-	0.24	0.36	0.48	0.59	4 / 3.7
	pick-3	-	-	0.34	0.46	0.57	4 / 4.0
	TSA	0.13	0.24	0.34	0.44	0.53	4 / 4.1
	SWA	-	0.24	0.34	0.44	0.53	4 / 4.1
	LP	0.13	0.26	0.40	0.53	0.66	3 / 3.3
	SDP	0.13	0.26	0.40	0.53	0.66	3 / 3.3
$32 \times 40$ ( $\rho = 0.8$ )	pick-1	0.09	0.18	0.27	0.37	0.46	5 / 5.0
	pick-2	-	0.17	0.26	0.34	0.43	5 / 5.0
	pick-3	-	-	0.25	0.33	0.42	5 / 5.0
	TSA	0.09	0.17	0.25	0.32	0.39	5 / 5.0
	SWA	-	0.17	0.25	0.32	0.39	5 / 5.0
	LP	0.09	0.18	0.27	0.37	0.46	5 / 5.0
	SDP	0.09	0.18	0.27	0.37	0.46	5 / 5.0

<sup>a</sup>  $\rho = m/n$     <sup>b</sup> Linear Programming [16]    <sup>c</sup> Semidefinite Programming [9]

<sup>d</sup> Average of the largest recoverable sparsity from 10 samples

Table A.8: Operation time (Fourier Matrix) - Average from 10 samples

Unit: minute (Rounded off to the nearest hundredth)						
matrix $A$ ( $m \times n$ )	Algo.	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$
$20 \times 40$ ( $\rho^a = 0.5$ )	pick-1	0.10	0.10	0.10	0.10	0.10
	pick-2	-	3.39	3.39	3.39	3.39
	pick-3	-	-	83.95	83.95	83.95
	TSA	0.10 <sup>d</sup>	3.40 <sup>e</sup>	7.51 <sup>e</sup>	82.91 <sup>f</sup>	229.10 <sup>f</sup>
	SWA	-	6.71 <sup>a</sup>	35.61 <sup>b</sup>	90.81 <sup>c</sup>	117.87 <sup>c</sup>
	LP <sup>g</sup>	0.27	0.01	0.01	0.01	0.04
	SDP <sup>h</sup>	0.62	42.14	49.40	45.80	44.62
	ESM <sup>i</sup>	0.09	3.60	91.29	1.689e3	2.4321e4
$24 \times 40$ ( $\rho = 0.6$ )	pick-1	0.12	0.12	0.12	0.12	0.12
	pick-2	-	3.30	3.30	3.30	3.30
	pick-3	-	-	84.84	84.84	84.84
	TSA	0.17 <sup>d</sup>	3.45 <sup>e</sup>	5.27 <sup>e</sup>	15.51 <sup>e</sup>	141.43 <sup>f</sup>
	SWA	-	6.73 <sup>a</sup>	8.42 <sup>b</sup>	88.43 <sup>c</sup>	96.84 <sup>c</sup>
	LP	0.01	0.01	0.02	0.01	0.03
	SDP	0.62	35.35	37.38	38.85	38.28
	ESM	0.09	3.62	91.70	1.697e3	2.4430e4
$28 \times 40$ ( $\rho = 0.7$ )	pick-1	0.09	0.09	0.09	0.09	0.09
	pick-2	-	3.35	3.35	3.35	3.35
	pick-3	-	-	86.77	86.77	86.77
	TSA	0.13 <sup>d</sup>	3.44 <sup>e</sup>	5.88 <sup>e</sup>	50.50 <sup>e</sup>	105.85 <sup>f</sup>
	SWA	-	5.97 <sup>a</sup>	12.23 <sup>b</sup>	94.89 <sup>c</sup>	223.50 <sup>c</sup>
	LP	0.01	0.01	0.01	0.01	0.02
	SDP	0.56	29.06	34.48	37.15	36.29
	ESM	0.09	3.66	92.77	1.716e3	2.4713e4
$32 \times 40$ ( $\rho = 0.8$ )	pick-1	0.10	0.10	0.10	0.10	0.10
	pick-2	-	3.51	3.51	3.51	3.51
	pick-3	-	-	89.01	89.01	89.01
	TSA	0.15 <sup>d</sup>	3.66 <sup>e</sup>	4.48 <sup>e</sup>	6.65 <sup>e</sup>	28.56 <sup>e</sup>
	SWA	-	6.49 <sup>a</sup>	7.66 <sup>b</sup>	92.00 <sup>c</sup>	95.13 <sup>c</sup>
	LP	0.01	0.10	0.01	0.01	0.01
	SDP	0.61	29.53	30.63	32.99	40.45
	ESM	0.09	3.69	93.58	1.731e3	2.4930e4

<sup>a</sup> Sandwiching Algorithm based on the pick-1-element algorithm<sup>b</sup> Sandwiching Algorithm based on the pick-2-element algorithm<sup>c</sup> Sandwiching Algorithm based on the pick-3-element algorithm<sup>d</sup> 1-Step TSA    <sup>e</sup> 2-Step TSA    <sup>f</sup> 3-Step TSA<sup>g</sup> Linear Programming [16]    <sup>h</sup> Semidefinite Programming [9]<sup>i</sup> Exhaustive Search Method (Estimated Operation time = average time to solve (2.9) for an index set from 100 samples  $\times$  total number of index sets)